Technical Report

UCAM-CL-TR-669
ISSN 1476-2986

Number 669

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Active privilege management for distributed access control systems

David M. Eyers

June 2006

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted June 2005 by the author for the degree of Doctor of Philosophy to the University of Cambridge, King's College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

Abstract

The last decade has seen the explosive uptake of technologies to support true Internet-scale distributed systems, many of which will require security.

The policy dictating authorisation and privilege restriction should be decoupled from the services being protected: (1) policy can be given its own independent language syntax and semantics, hopefully in an application independent way; (2) policy becomes portable – it can be stored away from the services it protects; and (3) the evolution of policy can be effected dynamically.

Management of dynamic privileges in wide-area distributed systems is a challenging problem. Supporting fast credential revocation is a simple example of dynamic privilege management. More complex examples include policies that are sensitive to the current state of a principal, such as dynamic separation of duties.

The Open Architecture for Secure Interworking Services (OASIS), an expressive distributed role-based access control system, is traced to the development of the Clinical and Biomedical Computing Limited (CBCL) OASIS implementation. Two OASIS deployments are discussed – an Electronic Health Record framework, and an inter-organisational distributed courseware system.

The Event-based Distributed Scalable Authorisation Control architecture for the 21st century (EDSAC21, or just EDSAC) is then presented along with its four design layers. It builds on OASIS, adding support for the collaborative enforcement of distributed dynamic constraints, and incorporating publish/subscribe messaging to allow scalable and flexible deployment. The OASIS policy language is extended to support delegation, dynamic separation of duties, and obligation policies.

An EDSAC prototype is examined. We show that our architecture is ideal for experiments performed into location-aware access control. We then demonstrate how event-based features specific to EDSAC facilitate integration of an *ad hoc* workflow monitor into an access control system.

The EDSAC architecture is powerful, flexible and extensible. It is intended to have widespread applicability as the basis for designing next-generation security middleware and implementing distributed, dynamic privilege management.

To Michael, Kerrie and Rachel

Acknowledgements

I owe a great deal to many people for helping me complete this thesis. Firstly, thanks to my supervisor Dr Ken Moody. Beyond enabling me to begin study at Cambridge, he has provided invaluable guidance, advice, proof-reading and all sorts of interesting opportunities over my PhD years. He had an uncanny knack of stepping in at just the right moment to keep me directed and happy with my work, and was also very supportive of my many and varied extra-curricular distractions. I also express my thanks to Dr Jean Bacon, who has provided a great deal of further advice, feedback and highly enjoyable collaborative research projects. A particularly special thank you to my mother, Kerrie Eyers, for keeping up her record of proof-reading my dissertations.

There are many other Opera Research Group members who have made my time at the lab enjoyable and educational. To András, Alan, Brian, Lauri, Nathan, Peter, Walt, and Wei, I thank you for our collaborations. I further thank you for greatly appreciated support and friendship, along with Andy, Aras, Chris, David, and Eiko.

Thanks also to King's College. I am proud to have become a member, and have gained wonderful experiences through singing, teaching, dining, socialising, rowing and living in its halls.

Finally, I thank my family. Although we have been mostly separated by thousands of kilometres, your support, love and encouragement has been an ever-present guiding light.

List of Publications

- Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. An asynchronous rule-based approach for business process automation using obligations. In *Proceedings of the Third ACM SIGPLAN Workshop on Rule-Based Programming (RULE'02)*, Pittsburgh, USA, October 2002.
- Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. A coverage-determination mechanism for checking business contracts against organizational policies. In *Proceedings of the Third VLDB Workshop on Technologies for E-Services (TES'02)*, 2002.
- Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. Mechanical consistency analysis for business contracts and policies. In *Proceedings of the Fifth International Conference on Electronic Commerce Research*, Montreal, Canada, October 2002.
- Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. Compliance checking for regulated communities. In *Proceedings of the 5th CaberNet Plenary Workshop*, Porto Santo, Madeira Archipelago, Portugal, November 2003.
- Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. Regulating web-based communities. In *In Proceedings of the IADIS International Conference on Web Based Communities (WBC2004)*, Lisbon, Portugal, March 2004.
- Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. An event-based paradigm for e-commerce application specification and execution. In *Proceedings of the 7th International Conference on Electronic Commerce Research (ICECR7)*, Dallas, Texas, June 2004.
- Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. Towards a benchmark for e-contract checking and monitoring. In *Proceedings of the 7th International Conference on Electronic Commerce Research (ICECR7)*, Dallas, Texas, June 2004.
- Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. Practical contract storage, checking, and enforcement for business process automation. In Steven O. Kimbrough and D.J. Wu, editors, *Formal Modeling for Electronic Commerce: Representation, Inference, and Strategic Interaction*, pages 33–77. Springer-Verlag, 2004.
- András Belokosztolszki and David M. Eyers. Shielding RBAC infrastructures from cyberterrorism. In *Research Directions in Data and Applications Security*, pages 3–14. Kluwer Academic Publishers, 2003.

- András Belokosztolszki, David M. Eyers, and Ken Moody. Policy contexts: Controlling information flow in parameterised RBAC. In *Policy 2003: IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, 2003.
- András Belokosztolszki, David M. Eyers, and Ken Moody. A formal model for hierarchical policy contexts. In *Policy 2004: IEEE 5th International Workshop on Policies for Distributed Systems and Networks*, pages 127–136, June 2004.
- András Belokosztolszki, David M. Eyers, Peter R. Pietzuch, Jean M. Bacon, and Ken Moody. Role-based access control for publish/subscribe middleware architectures. In *International Workshop on Distributed Event-Based Systems (DEBS03)*, 2003.
- András Belokosztolszki, David M. Eyers, Wei Wang, and Ken Moody. Policy storage for role-based access control systems. In *Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03)*, 2003.
- Nathan Dimmock, András Belokosztolszki, David M. Eyers, Jean M. Bacon, and Ken Moody. Using trust and risk in role-based access control policies. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 156–162. ACM Press, 2004.
- David M. Eyers and Ken Moody. Credential negotiation with limited disclosure via iterative range refinement in an unordered space. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, page 427. IEEE Computer Society, 2003.
- David M. Eyers, John Shepherd, and Raymond Wong. Merging Prolog and XML databases. In *Proceedings of the Seventh Australasian Document Computing Symposium (ADCS'02)*, December 2002.
- Steve Neely, Xiaofeng Gong, David M. Eyers, Julian Newman, Helen Lowe and Jean M. Bacon. A tutorial task and tertiary courseware model for collaborative learning communities. *Electronic Journal on E-Learning*, 2(1):159–166, March 2004.
- Steve Neely, Helen Lowe, David M. Eyers, Jean M. Bacon, Julian Newman, and Xiaofeng Gong. An architecture for supporting vicarious learning in a distributed environment. In *Proceedings of the 2004 ACM symposium on applied computing*, pages 963–970. ACM Press, 2004.

Contents

List of Figures

List of Tables

1 Introduction

The last decade has seen extraordinary growth in the number of distributed systems based on World Wide Web technology. Accompanying this growth has been the increased need for distributed access control architectures, as the web has developed from a public information delivery medium to one supporting personalised delivery of services. These access control architectures range from simple authentication systems to complex policy-based authorisation systems. At either end of this complexity spectrum, to ensure safe operation, access control architectures should provide mechanisms to cope with change in credential status.

Naturally, moving from a locally-contained system to the distribution of access control requires some sort of communication protocol between sites which 'know' the validity of a credential and the sites which want to perform actions requiring authorisation. Traditionally, there have been two main approaches. One approach is to send queries back to the authoritative credential issuer at the time of usage to confirm validity. Such approaches suffer a large performance penalty, however, and do not scale well – in the limit they will place undue load on the aforementioned authoritative credential issuer.

The other main type of approach has been to employ a technology such as Public Key Infrastructure (PKI – [HFPS99]) to allow local sites, via cryptographic algorithms, to verify credentials through the sites' digital signatures. Because the cryptography is assumed to be unforgeable, the site of usage can check back through a linked chain of certificates to some trusted root certification authority.

Applications using this sort of approach mostly assume infrequent change to the validity of signed credentials. The X.509 certificate standard (one of the most common forms of PKI currently employed) includes a mechanism for distributed publication of Certificate Revocations Lists (CRLs). However, this approach is still predominantly a 'pull' model in current deployments, meaning that a server desiring to verify a particular authorisation needs to explicitly check the CRLs if they fear that certificate validity may have changed. Particularly problematic is that many X.509 application frameworks lull software developers into a false sense of security – it is assumed credentials are valid, when in fact the underlying certificate library is not actually checking the CRLs as exhaustively as it should. There is a disadvantage in the certificate management code and the network protocols which actually facilitate certificates being sent through the network being mutually agnostic: the certificate management policy cannot specify a coherent course of action should network services become unavailable (due to attacks from hackers, for example).

In contrast, the Open Architecture for Secure Interworking Services (OASIS – see [Hay96], but also [BMY01, YMB01, HYBM00, BMB$^+$00]) has, since its proposal in the mid 1990s, suggested usage of a 'push' model for credential status update. By setting up event channels, a service relying on validity of a credential may register interest in that credential's status changing. Thus we get the benefits of fast response in the face of credential status change, without the inconvenience and cost of having to constantly 'poll' credential status. At a lower protocol level, we also gain an insight into the status of network connectivity. Further, incorporating credential update into the actual access control protocol avoids the risk that different servers choose to implement their polling in inconsistent ways (perhaps one polls on every credential usage, where another just polls CRLs once an hour).

The work behind this thesis, however, represents the first success at actually implementing this active messaging support into a demonstrable OASIS system. Indeed, the reason it was not successfully implemented previously was because the original OASIS design did not provide a specific scalable approach to message delivery. Significant advances have been made in addressing this scalability issue.

This thesis presents a new access control architecture, *Event-based Distributed Scalable Authorisation Control* (or EDSAC21 – often referred to as EDSAC). The principles on which EDSAC is built include those used in the OASIS project, but further, include support for dynamic constraints and other policy features. The EDSAC model is a multi-layer architecture with replaceable components. We aim to unify many different potential access control components by specifying how they interact rather than by presenting one particular feature-set. Focusing on these components is intended to remove much of the diffuseness of the earlier OASIS designs, where key parts of a distributed access control architecture were described, but other parts were left ambiguous, leading earlier OASIS implementations to differ significantly in their interpretation of the core model.

Beyond making clear our shift to a more compartmentalised model, it is also necessary to note that OASIS now shares an acronym with the Organization for the Advancement of Structured Information Standards (i.e. OASIS – [OAS03]). This has overloaded the name significantly, particularly in that their charter of releasing open standards includes access control technologies too (notably XACML [OAS02]). Rather than searching for a new acronym, it seemed a better solution to overload a name which has already been of great use and historical significance to the Cambridge Computer Laboratory.

1.1 Active, distributed privilege management

This section explores distributed access control environments in which entities communicate privilege state using 'push' style communication. The authoritative owner of a credential, or their delegate, must take explicit action to effect notification of all interested parties when the status of a credential changes. For this reason, we describe this style of privilege management as 'active'. It also implies that we incorporate a message delivery infrastructure within our access control system. Throughout the rest of this thesis these messages may also be referred to as 'events'.

Providing a scalable messaging infrastructure within a distributed access control framework facilitates many policy features in addition to fast revocation, however. The ability to react to changes in the state of the access control system allows other dynamic constraints to be checked. For example, it is possible to ensure dynamic separation of duties between sets of credentials. In such a situation, if a principal requests to enter a role which is specified to be mutually exclusive with another role, the currently active role holder can instruct the requester to drop their request, or can deactivate their own role in response.

When enabled to handle dynamic constraints, access control systems such as EDSAC can be extended to provide workflow support. Basic workflow specifies a directed task dependency graph in which each node has certain privileges associated with it, and edges can only be traversed if the condition associated with them is satisfied. It also allows the association of multiple instances of a workflow with a given principal (generally access control methodologies only associate a single state with each principal). At any point in time for a particular instance of a workflow, a principal will sit at a given node in the task dependency graph, and will perform their work based on the privileges currently accessible to them from that node.

Large-scale distributed access control systems will often be deployed in an environment where an audit log is legally required for future accountability. Another use of an active access control messaging infrastructure is to facilitate audit servers that signal their interest in recording certain events for their records. Of course, attention must be paid to the security of the connections to such servers and their storage.

To some degree, audit logging is not necessarily an 'active' application in terms of privilege management, however we feel an active approach is much more reliable and convenient. Independent servers could potentially maintain their own audit logs to be combined into the overall audit at a later date, but this in itself raises the question of each site's accountability. Glitches in server behaviour may also complicate this data integration.

Active privilege management also permits policy administrators to take a more direct role in access control system operation. For example, we explore an application in which the current status of a particular role activation is displayed. This allows an administrator, or indeed the original principal themselves, to discover why a particular role activation request may not be progressing as they expected. It is advisable for such functions to operate, to a degree, in a reflexive manner, meaning that the privilege to examine the access control system state is itself a privilege managed by the access control system.

Finally, also related to the administration of the access control system, is management of policy evolution. Many existing distributed access control infrastructures assume that policy changes are infrequent, and thus that the system would need per-site intervention to update them, or assume that local policy caches are valid for some set period of time before refreshing. Active messaging infrastructures can increase the responsiveness of distributed access control systems to (potentially critical) policy changes, as well as providing a mechanism for external policy administration. Again, policy modification privileges should be managed by the access control system itself.

1.2 Research issues

The ideas presented in the previous section raise a number of important research problems which are outlined below.

Security. The first big research issue revolves around how to balance the desire to distribute systems for flexibility whilst maintaining sufficient security over their operations. At the low-level (that is, the system level), there are issues relating to whether the access control infrastructure can itself come under attack. At higher levels (policy design), potentially complex policy specifications may lead to security holes accidentally being included into the access control rules. This is a particularly risky problem when working with the privileges which themselves allow further modification of policy, or with external predicate calls whose semantics are not fully understood. Beyond managing design flaws in policy, there may also be risks associated with the level of trust given to administrators, or indeed servers in a network.

Security is an even more significant concern when proposing active privilege management via an event delivery infrastructure. If incorrectly protected, sensitive information about the behaviour of principals may be mistakenly disseminated to unauthorised sites. Our research proposes a highly interconnected infrastructure, thus we need to solve the problems caused by intermediaries necessarily passing messages throughout the distributed access control system.

Scalability. Developing an architecture for wide-area many-to-many messaging with flexible participation which scales well is non-trivial. We employ publish/subscribe research, however such messaging systems generally do not have a focus on security, instead concentrating more on the efficiency of message delivery. Thus most publish/subscribe models fall short of our security requirements. It is not a problem if message delivery is slowed slightly if it permits us to keep track of exactly where message copies have (and have not) been delivered.

Reliability. Unlike many other publish/subscribe messaging infrastructures, highly reliable message communication is essential. This means that we need a method for determining the continued inter-server connectivity, and performing appropriate recovery actions at any site if it appears to have become isolated from the access control network. At the same time there must be a balance with scalability needs, so as not to arrive at a situation in which a large proportion of network bandwidth is consumed merely for the sake of testing continued connectivity.

Manageability. The final challenge is in designing a manageable system. That is, one which can be readily understood by users, and safely managed by administrators. After all, the primary goal of policy-based access control should be to allow users a clear view of the explicit policy rules which would otherwise be implicitly encoded into a given application's code. Over-complicating an architecture will lead to low-level flaws, similarly over-complicating the policy language will lead to higher-level rule specification flaws.

1.3 Thesis contribution

The primary contribution of this thesis is the specification and prototype implementation of an architecture for highly-scalable active privilege management. This involves both system-level design, and policy specification language extensions. Using load sharing over a network of interconnected access control servers, the EDSAC architecture facilitates highly responsive revocation, and allows efficient, collaborative determination of dynamic constraints. A number of other related contributions of this thesis are described below:

- Presenting a clear, multi-layer, component-based framework in which to implement the EDSAC model. We fully recognise that different uses of EDSAC will require varying amounts of access control infrastructure, as represented in the EDSAC layers. Further, certain deployments may need to change components of the system, hence our focus on individual EDSAC components and how they interact. For example, deployments might choose to change the role-aware policy language being used, the authentication system, or the database technology used to store policy or user session state.
- Proposing a specific event-type set to facilitate the active, scalable management of access control credentials within a large-scale distributed system. These event-types sit within a topic- and attribute-based publish/subscribe messaging infrastructure.
- Using a heartbeat protocol to bound the acceptable network delay in event propagation. This allows some aspects of the EDSAC protocol to operate in a synchronous manner. All EDSAC nodes are able to reason about their own connectivity, and take appropriate security precautions (for example the revocation of roles), if they assess they have become disconnected from the network. We also discuss how a node re-joins the network, given that any period in which privilege management messages have been missed may have caused this node's state to become desynchronised from the rest of the access control system.
- Advancing publish/subscribe research by the addition of access control to such systems. Given that a publish/subscribe messaging system is employed to support our EDSAC distributed access-control framework, it seemed logical to expose publish/subscribe services to applications alongside our access control services.
- Developing the notion of *policy contexts* for the grouping of OASIS roles, rules and parameters of both. Contexts allow us to check certain information flow constraints at policy deployment time, and are used by the EDSAC prototype to implement dynamic constraints. Policy contexts also provide a useful tool for grouping policy

elements for analysis, navigation, and making these elements the target of higher-level policies.

- Extending the OASIS policy language into that used for our EDSAC prototype. We add support for membership conditions and other dynamic constraints based on policy contexts. Policy contexts are shown to conveniently model parallel dependency paths in workflow specifications.

- Presenting a two-phase role-activation protocol necessary to reduce undesired side-effects in dynamically-sensitive policy systems. Although dynamic constraints greatly increase the flexibility of access control systems, activation of a role may have side-effects in other parts of the distributed system. This presents a particular problem for role-activation techniques which automatically guide a user, when provided with a set of credentials, to a target role, since such systems may need to 'test-activate' a role to see whether it can reach the privileges the principal desires. In a dynamic policy system, 'test-activation' of a number of roles might cause excessive, unnecessary, and possibly damaging revocations. Beyond that immediate concern, the audit logs would also grow rapidly, and it would be very hard to discern where role revocations were accidental side-effects versus when severs where, in fact, correctly making the decision to preserve security.

  Our two-phase role-activation approach allows a system to plan a particular course of action without undue side-effects. The second phase attempts to activate the given role in an atomic manner. Information about what phase of operation actions are performed can be recorded in the audit logs, since both phases still propagate privilege management messages. Note that we assume that knowledge about potential side-effects is not sensitive; policy authors should ensure that their design will still be secure even if a hijacked EDSAC node was harvesting side-effect information.

- Developing support for workflow specification via credential exclusion groups and two-phase role activation. We show that the active privilege management support in EDSAC provides the basis on which we can develop an *ad hoc* workflow monitor. We discuss how the use of computer-controlled principals can implement obligation policies within EDSAC.

- Demonstrating policy visualisation as another client of active privilege management. While this thesis does not focus on human-computer interaction research, we feel that graphical approaches to policy management are certainly worth exploring, if only as a parallel track to the existing textual presentations. We present the tools we have developed to translate OASIS and EDSAC policy files into coloured graph representations.

  Our initial work was on static graph presentations, in which we show the flow of required credentials to reach a target role. Because this analysis was static, we can only assume that the semantics of predicates on the path from initial to target roles are understood by the principals who will need to use them for role activations.

We then describe how we extended our approach by using the underlying privilege management architecture to show dynamic policy state. Thus, if suitably authorised, a principal can request a graphical display showing the progress of some recent role-activation request. Unlike in the static representation, the graph clearly shows which conditions have evaluated to be true or false, including environmental predicate call-outs from the OASIS and EDSAC system.

The thesis of this dissertation is that the integration of a content-based message delivery infrastructure into OASIS (or other suitable decentralised role-based access control systems) realises an efficient, simple, and distributed mechanism for the checking of a powerful and expressive set of dynamic policy constraints.

1.4 Dissertation outline

The structure of this thesis is as follows:

Chapter 2 provides the research background to distributed access control and scalable messaging. We provide an overview of access control models leading up to the Role-Based Access Control (RBAC) on which OASIS (and subsequently EDSAC) are based, including discussion of the types of policy supported by each. The challenges presented by distributing access control are then detailed.

Many of the main contributions of this thesis rely on the use of scalable messaging infrastructures. We thus also introduce publish/subscribe systems, and describe the Hermes topic-based publish/subscribe system used in our prototype implementation.

Chapter 3 presents a range of principles we consider to be crucial in developing real-world distributed access control systems. These principles are based on our experience in deploying OASIS, as well as on recent developments by other researchers.

Chapter 4 discusses OASIS and its implementations in detail. The OASIS project has been active now for nearly a decade, and has seen many researchers and implementations come and (unfortunately) go. We describe the current CBCL OASIS implementation, and contrast it with previous designs.

Chapter 5 reports our experiences in deploying OASIS in two experiments. The first involved building a prototype Electronic Health Record (EHR) system for the United Kingdom National Health Service (NHS). The second used OASIS in a courseware system supporting inter-organisational collaboration.

Chapter 6 introduces the EDSAC model and describes its various layers. We discuss the components which participate in each layer, and the events used to facilitate its policy features. We also describe the EDSAC two-phase role activation protocol and indicate how role-aware policy languages integrate with it.

Chapter 7 examines our EDSAC prototype implementation, and demonstrates its ability to handle dynamic constraints. The case studies we provide include using EDSAC within location-aware access control systems, and an example workflow system. Our workflow example shows EDSAC allowing collaborative completion of workflow tasks shared between different principals, whilst upholding dynamic constraints based on valid workflow progression.

Chapter 8 concludes this thesis, providing a summary of its achievements, difficulties encountered, and recommendations for directions in which future research may proceed based on the ideas presented.

2 Related work

This chapter examines the past research on which this thesis is dependent. Note that there are two mostly separate sections of related work; the first and larger area is that of computer security (§2.1) and particularly access control research (§ 2.2). Section 2.3 introduces related research in the field of scalable messaging systems.

This thesis is focused on access control, however we also present a quick overview of scalable messaging systems. Our active security infrastructure requires a scalable underlying message delivery system in order to be scalable itself.

2.1 Computer security

Computer security is an increasingly studied field given the proliferation of large-scale distributed systems which run on the Internet. The main challenges have stemmed from an increase in the number and power of computers, coupled with a dramatic increase in the complexity of their operating system software. The situation has been further exacerbated by a dramatic rise in the degree of their interconnection. For good, high-level overviews of computer security, see [And01, Gol99].

Only a few decades ago, computer security was a single-machine task – a system administrator would allocate privileges to users, and monitor the system for abuses. Frequently users would use dedicated communication lines to reach the shared computing resources, making this monitoring job easier – access was only possible in limited, physical ways.

Today, however, a vast number of computers are permanently connected to network infrastructure. Our operating systems run a bewildering range of (possibly incorrectly implemented) background services. Network communication, and by definition the interaction with other computing devices, is often managed entirely transparently to the user of a computer. Anyone who runs a Microsoft Windows operating system on their computers will know security updates are released frequently to patch holes in critical software components. Table 2.1 shows the frequency of bulletins released over the past five years [Bek03]. Note that each bulletin will fix an undisclosed number of actual security flaw instances. Microsoft has begun to distribute automatic updates prior to releasing public notifications, since malicious programmers have proven they can develop software to

Year (or part thereof)	Number of security bulletins issued
2004 (to June)	17
2003	51
2002	72
2001	60
2000	100
1999	61

Table 2.1: Recent annual frequencies of Microsoft security update bulletins

exploit a vulnerability within days of hearing about it.

Security holes in widely-deployed network infrastructure are now routinely used both for the replication and transmission of malicious Internet 'worms'. Due to increasingly wide-spread Internet availability, these parasitic program entities can often be remotely coordinated, and their hosts (so called 'zombies') instructed to participate in unsolicited email distribution, or 'distributed denial of service' attacks. The latter involves a team of zombie computers each creating a sufficiently saturated communication stream to some network location that their combined effect disables it. These vulnerabilities relate to computers interacting with each other via their operating system software.

Protecting against human weaknesses in a security system is also a significant requirement. Usually this is a user education concern. For example, users should know why it is that certain requirements are made of their password formation, and why it is important to honour mandated password rotations. Otherwise they may be tempted instead to try to increase their own convenience by finding ways around their administrators' security policy.

2.1.1 Defining computer security

Before describing aspects of research within such a broad field, we should first attempt to define computer security. Our working definition is as follows:

> "Computer security refers to the mechanisms employed to monitor and protect electronic resources."

In the non-electronic world, security generally relates to the protection of assets. We use 'resources' to indicate that not only computer assets, i.e. data, but also computer services are to be protected. We also include the notion of 'monitoring' to indicate that security is not only about attempting to prevent unauthorised access to resources, but also involves mechanisms to detect unauthorised usage after such an event has occurred. The information gained by such monitoring activities can then be fed back into the protection mechanism to enforce new conditions, or specific sanctions against a detected offending entity.

If we focus on data, computer security is generally accepted to cover three main areas to avoid this data being compromised:

Confidentiality. Maintaining confidentiality implies ensuring unauthorised entities do not gain access to information they should not see. There are two main fields of security research which deal with confidentiality. The first is cryptography. If a cryptographic algorithm is sufficiently secure, data is confidential except for those who have an unlocking key. The second main field of research is access control, which often itself uses cryptography to effect its aims. The focus of access control is on who is allowed to see data.

Integrity. Integrity involves ensuring that unauthorised entities do not modify protected data. Note that in the worst case a breach of this type will also involve a breach of confidentiality (i.e. unauthorised read followed by data modification). This may not necessarily be the case, however – a security breach may involve the blind destruction of protected data. Again, both access control and cryptographic research tools are employed to manage this security aspect. Access control should ensure that only authorised entities can modify protected data. Cryptography provides us with two related technologies; message authentication codes (MAC) and digital signatures.

Message authentication codes use a one-way function to convert protected data into a short (usually fixed-length) summary code. It is impractically expensive in computing terms to try to generate source blocks which map to a particular MAC. To turn this MAC into a signature we can use either symmetric or asymmetric cryptography. In the symmetric case, the sender and receiver must share a secret. The sender appends this secret to the message before generating the MAC. The receiver does the same, and knows that only malicious entities who also know the shared secret could modify the message and regenerate a valid signature. Another option is to use asymmetric cryptography. In this case, the sender encrypts the MAC with their private key, knowing that anyone with the public key can verify that the MAC is valid, but that only attackers with the private key can regenerate the MAC on a modified message.

Availability. Finally, there is the requirement that authorised entities in the system should be able to access protected resources whenever they so desire. Access control research plays a part, since it should not block access incorrectly. Such access requires that users can actually reach the access control gateway behind which the protected resource lies, however. Thus attackers can target the communication infrastructure itself, e.g. through distributed denial of service (DDoS) attacks. The security research to combat this threat involves either replication of the resource access points to avoid the problem, or at least ensuring rapid recovery after attacks have caused temporary failures.

If we look instead at the protection of software services, the above ideas apply, but require some renaming to make sense – for data we refer to protected state, whereas there is possibly no state stored on the computer from which we might request a service. The notion of confidentiality thus corresponds to the principle of ‘accessibility’, namely that only authorised entities are permitted progress with requests made to a service. The term 'validity' better represents service integrity: that the results of a request are not interfered with on their way back to the requester. The notion of 'availability' applies as for data.

2.1.2 Low-level security tools

This section identifies a number of key security technologies on which we build the EDSAC access control infrastructure, but on which our research is not directly focused.

Cryptography

Whilst we often use cryptographic tools in this thesis, and discuss cryptography in our designs, we make no contributions to the field itself (see [CB02] for a good case study into breaking cryptographic security on smart cards). We instead assume that reliable cryptographic tools exist that we can apply in a black-box manner. This is a common approach for network-level security from the application perspective – for many years cryptographic tools have been available with the Java Cryptography Extensions (JCE). These tools have recently been integrated directly into the Java run-time system. Equivalently, Secure Sockets Layer (SSL) routines are often deployed from an implementation such as OpenSSL, without programmers knowing exactly how the various algorithms involved work (nor the specifics of the security protocol handshaking between communicating entities).

The particular cryptographic concepts we need are described below:

Symmetric cryptography. Symmetric cryptography involves use of a bijective function which maps data from a source to a destination representation. The mapping applied by this function will depend on a symmetric key provided to it. The degree of security provided by the function is determined by how easily the source representation can be derived from the destination representation. Hopefully this process is computationally impractical to an interloper, even if the mechanics of the cryptographic function are known.

Predictably, this cryptography is called 'symmetric' because the same key is used to encrypt and decrypt data.

Asymmetric cryptography. Asymmetric cryptography uses one mapping function for encryption and one mapping function for decryption. Unlike symmetric cryptography, the keys used for each function are different. After a key-pair is generated, the usual methodology applied is to keep one half private, and provide the other publicly to any interested party. This allows a particularly powerful basis for signatures, since an entity can distribute a message which includes a MAC encrypted with their private key. Assuming any receiver has access to the sender's public key, the message can have its MAC calculated by the receiver, and then the sender's public key applied to decrypt the sender's MAC. If these two MACs match, a receiver, possibly unknown to the sender, has confirmed that the message really belongs to that sender.

It is important to realise that the distribution of public keys is potentially subject to "man in the middle" attacks. If a malicious party is able to intercept a receiver's request for a public key, it can replace the sender's public key with its own and decrypt and re-encrypt messages in both directions. However, once a given point of trust is correctly established between two communicating entities, they can use this trust to vouch for the validity of further public key dissemination through the use of digital signatures.

For most e-business Internet applications, this process of establishing that a client is interacting with a given server is simplified by web browsers shipping with preloaded trusted root certificate authorities (CAs). Organisations pay subscription fees to these authorities to have their credentials digitally signed, and thus are able to be verified by the client browsers. Examples of such authorities at the time of writing include VeriSign, Entrust, Thawte and many others. Organisations may also provide certification authorities of their own, for example the Cambridge University Card Centre is the trust authority for information contained in this university's students' smart-cards.

X.509 certificates and SSL/TLS.

Digital certificates are cryptographically signed credentials. Usually we will use 'digital certificate' or 'certificate' to mean 'public-key certificate', which implies that the signed credential, among other attributes, contains a public key.

Certificates consist of a well-structured description of a credential, which is signed via asymmetric cryptography. A root certificate will have self-signed credentials. Otherwise, some higher-level authority will sign your credentials with their private key. Anyone who has access to this authority's public key can check this authority's verification of your claim. The Secure Sockets Layer (SSL), or almost identical but newer IETF standard Transport Level Security (TLS) [Die99], protocols define how two parties can establish a secure channel for communication. For such channels to be useful for security purposes, at least one end of this communication must be identified with a digital certificate.

These protocols are highly flexible, allowing negotiation of a variety of different encryption and MAC algorithms, as well as such features as compression of transmitted data. HTTPS, which is the HyperText Transfer Protocol (HTTP) transmitted over SSL, for e-business usually involves an unknown client connecting to a certified server (e.g. an Internet banking site). Having established a secure point-to-point link, the banking site will then establish client identity, although this is currently through an *ad hoc* process – for example entering login, password and correctly answering a challenge question to which only the real user should know the answer.

The attributes within the X.509 certificates [HFPS99] used in SSL/TLS [Die99] follow the X.500 naming scheme, and are structured using the Abstract Syntax Notation (ASN.1). As the name suggests, ASN.1 does not actually specify an encoding form – this is done by a set of Definite Encoding Rules (DER). The concept behind this separation is that one object may be encoded using different DERs, for example, a compact binary representation versus a human-readable format.

When the term ‘X.509 certificate’ is used, it usually refers to version three of the standard. X.509 Version 1 was released in 1988, and meets most basic certificate needs, but lacks the extensibility supported by version three. Version 2 was released in 1994, but did not gain wide-spread use. Version 3 was released in 1997. The most important feature of version three certificates is that anyone can define extension fields to be included within the certificate. An example of a common extension is the ‘KeyUsage’ field, which suggests what this certificate should be used for. Consider a user with two keys, one with double the bit-length of the other. The shorter bit-length (and consequently less secure) key may be permitted to sign documents, or initiate Secure SHell (SSH) connections, but may not be permitted for the use of signing other certificates – a function reserved for the more secure key. As we describe later in this thesis, we use X.509 extension fields to encode OASIS and EDSAC attributes within our digitally-signed credentials.

All X.509 certificates contain the following fields:

Version. This field specifies the overall structure of this certificate from the (currently) three X.509 versions.

Serial Number. Every certificate created must be assigned a serial number which should be unique for that issuer. Thus, when combined with the issuer name, we get a concise identifier for this certificate. These identifiers are used in Certificate Revocation List (CRL) entries, for example. CRLs are used to indicate that credentials, valid with respect to their digital signatures, have now been invalidated for other reasons.

Issuer Name. This field provides the X.500 name of the entity which signed this certificate. As discussed above, this will either be a certification authority, or the subject of this certificate, if it is self-signed. Trusting this certificate is predicated on trusting the issuer of this certificate, or someone who trusts them transitively. The term ‘certificate chain’ is used to describe the situation in which there are a number of steps from a certificate to its root certification authority through intermediate certification authorities. Certificate chains involve a sequence of certificates being presented; every certificate C_n is followed by the certificate C_{n+1} which signed C_n. This sequence terminates in the root certification authority certificate, which is self-signed. The end certificate C_0 can be trusted if any of the certificates in the chain is trusted.

The X.500 names of entities are intended to be globally unique, and are also called Distinguished Names (DNs). The issuer of the CBCL OASIS certificates is, for example:

```
C=UK, O=CBCL, OU=OASIS Primary Certification Authority
```

In this case ‘C’ indicates the country, ‘O’ indicates the organisation, ‘OU’ refers to the organisational unit. Distinguished names often also contain a common name (‘CN’) for the certificate.

Subject Name. In a similar manner to the issuer name, in this field each certificate will describe the name of the entity to whom this certificate has been issued. This name is also an X.500 DN. For example:

`C=UK, O=CBCL, OU=OASIS, CN=Doctor`

Note that in this case, attributes within the CBCL OASIS certificate identify which doctor this certificate is for, rather than this being part of the Certificate Name itself. This is indicative of the capability-based approach taken for OASIS appointments. This is discussed in detail in future chapters.

Validity Period. Any given certificate is only valid for a specific period of time. This is encoded via a start time-stamp (i.e. a structure including date and time) and an end time-stamp. It is thus possible to issue certificates that are not yet valid, as well as certificates with validity times ranging from seconds to hundreds of years.

Signature Algorithm Identifier. A number of asymmetric cryptography algorithms are available for signing the certificate Message Authentication Code. RSA, DSA, and ECDSA are provided specific OIDs in RFC 3279 [PHB02].

Subject Public Key Information. Public-key X.509 certificates also contain a public key, and a description of which cryptographic algorithm can interpret this key. Most algorithms will require some further parameters, e.g. to specify the key size for algorithms which support variable key lengths.

The above fields contain sufficient information to allow the validation of a certificate, assuming the validator has access to a trusted source certificate on the certificate chain, and any necessary cryptographic algorithms. However, sometimes certificates may need to be revoked before their expiry date. As certificates are self-contained packages, the only way to effect such revocations is to issue a revocation certificate. After checking any given certificate internally, X.509 software frameworks should always validate their certificates against Certificate Revocation Lists (CRLs). In practice, many applications do not validate CRLs automatically and in a timely manner.

This thesis proposes an alternative to frequently issuing and revoking certificates for dynamic elements of access control (e.g. roles) relying on communication connectivity instead of digital credentials. In some ways our approach is similar to the scheme proposed in [FPP$^+$02].

Human factors and computer security

This thesis does not focus on non-technologist end-user interactions with our computer security work. To do so, we would have needed to present our software interfaces to human testers, and thus determine whether they were able to administer security in a safe and efficient manner. Some of the case-study work presented in the latter parts of this thesis does explore user interfaces, but from the point of view of the technology required to build them, rather than performing a quantitative assessment of their comparative ease of use.

Those projects that did involve user interaction (particularly the Role-based Access-control for Evolution of Distributed courseware (RAED) project – see section 5.2), had computer science students as end-users – not a sufficiently representative social group from which to draw usability conclusions.

In summary, for the research leading to this thesis we generally assume that physical and electronic security can be established using existing tools, and do not directly consider the procedures an organisation would need to employ to control human agents with system administrative privileges from behaving maliciously.

2.1.3 Policy-based access control: terms and definitions

Our computer security research focuses on access control within distributed systems. By 'distributed systems', we mean that our users and services may be remote from each other. Access control most specifically addresses the confidentiality and availability aspects of the computer security definition we have given previously. We define the term as follows:

> "Access Control involves preventing unauthorised users from interacting with particular resources in certain ways, whilst guaranteeing that authorised users will not be denied their access rights."

Our use of the term 'interacting with resources' is purposely broad – we want to include reading, writing, appending, or any other appropriate mode of access. Some example interactions might include:

- Calling a particular method on a protected object instance,
- Interacting with operating system services,
- Invoking operations that actually create credentials,
- Issuing requests to revoke credentials,
- Submitting updates to a set of policy rules that will assess future authorisation.

These different interactions can all be facilitated by our OASIS and EDSAC models, because both architectures define access control systems that are decoupled from the resources they protect and/or the entities making requests for them. If EDSAC (or, without loss of generality, OASIS) is closely built into the operation of software at a particular site, it could be said to be providing 'security middleware'. Alternatively, if EDSAC services are operating at a location without any other application-specific resources, we can view EDSAC as providing 'security as a service'. The former term was appropriate for the CBCL OASIS implementation, and our RAED deployment.

The implementations of OASIS, and development of EDSAC are the foci of this thesis. They both provide the services required to name the entities in an application to be secured (e.g. protected objects and operations, and users of the system).

Another aspect of both EDSAC and OASIS is that they are policy-based. We provide our definition of policy:

> "Policy is a set of guidelines describing the intended behaviour of a system, and how to react to observed deviations from this behaviour."

Here we are using the term 'system' as an entity more general than computing environments. In non-computing environments, policy is often recorded in human-readable natural language. This language may well be structured into legal documents. Nonetheless, this definition is far too broad to be of use for access control within distributed computer systems. We thus provide a more focused definition of access control policy:

> "Access control policy is a collection of rules that define the conditions under which interactions may occur between requesting entities and the protected resources of a computer system."

When describing an access control system as being 'policy-based', we mean to emphasise that the rules by which a system assesses authorisation requests are decoupled from the access control monitor itself. In other words, the rules can be modified during operation without the access control infrastructure needing to be brought off-line, and then restarted.

Key terms

In most access control research, the entities which make access requests are referred to as *principals* or *subjects*. Usually, principals will associate fairly directly with human users of a computer system. However, there are many situations in which the computer-based processes performing an action should be considered as an access control principal, whether or not they are performing their actions on behalf of a specific human operator. For example, in a distributed system such as the Electronic Health Record experiment described in chapter 5, a human user may have a number of computer-based proxies performing any given request. The location and type of these proxies within the network may govern what policies apply to them.

When a principal is given access to perform some action on a resource, this is usually referred to as being granted a *privilege*. Often the privileges associated with a resource will depend on the mode of access for that resource. For example, permission to read a resource is often less security critical than the permission to write to that resource.

2.2 Access control research

Access control research predominantly focuses on the methods used to determine which principals can gain particular privileges. All such systems thus manage the paths between principals and privileges in some sort of graph structure. However, two broad categories of access control systems emerge, based on whether a strict ordering is also applied among the sets of principals and privileges.

In the cases without this strict ordering, we are in the class of Discretionary Access Control (DAC) systems. The notion of discretion comes from our ability to include *ad hoc* exception cases when necessary – perhaps a comparatively junior principal has the need for exceptional access to a highly privileged object, maybe during closely monitored training. DAC schemes can usually encode such exceptions.

Schemes that strictly order principals and privileges are known as Mandatory Access Control (MAC) models. They have the advantage of highly structured access control policy, and have behaviours circumscribed by systematic mathematical constraints. They are also highly inflexible. Non-technological applications of MAC can be seen in organisations such as the military, where the need to maintain hierarchy is more important than the desire for convenient flexibility.

It is important to note that MAC schemes can usually be defined within DAC models. Also, we provide Role-Based Access Control (RBAC) with its own section below, though it is in fact a type of discretionary access control. However, given that this thesis explores RBAC in detail, and contributes specific distributed RBAC implementations, we feel this treatment is justified.

2.2.1 Mandatory Access Control (MAC)

Before we introduce the most frequently used Mandatory Access Control models, we must first define the security lattice through which subjects and objects are compared.

A lattice is a structure of the form $(L, \leq)$, where L is a set and $\leq$ is a partial ordering such that for each pair of elements $a \in L$ and $b \in L$, there exists a least upper bound $u \in L$ and a greatest lower bound $l \in L$, such that the following conditions hold:

$$a \leq u, b \leq u$$
$$l \leq a, l \leq b$$

Further:

$$\forall v \in L, a \leq v, b \leq v \rightarrow u \leq v$$
$$\forall k \in L, k \leq a, k \leq b \rightarrow k \leq l$$

An alternative manner in which to visualise such structures is as an acyclic, directed graph, with nodes from the set L, and edges formed such that the partial ordering is preserved (see figure 2.1). The graph representing such a lattice can be topologically sorted – i.e. this type of graph can be drawn with all edges pointing in one direction, leading away from the absolute lower bound of the partial order, and leading toward the absolute upper bound.

For security purposes, the elements of set L (equivalently nodes in the graphical representation) are used to partially order labels which represent security classifications. In MAC schemes, both principals and target objects are tagged with security labels. A given scheme will define the permissible modes for which labelled principals may access labelled objects.

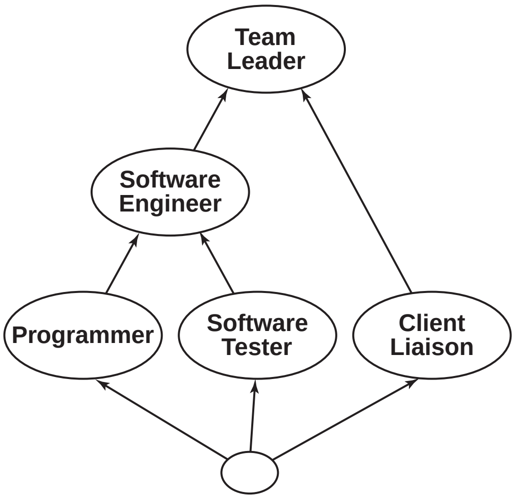


Figure 2.1: A security lattice

Bell-LaPadula Model

The Bell-LaPadula Model [BL73] provides a means to prove inductively that a system is secure. It relies on three basic rules; the *-property, the simple property and the tranquillity property. Note that it is built with attention to the semantics of information flow based on certain data-oriented access modes. For example, writing to a given object implies information flow to those principals capable of reading that object. We define each of the rules below:

***-property (star property).** A principal is only allowed to write to an object if the security level of the object is the same or greater than the security level of the principal. Thus information cannot leak out to principals with lower security clearances. For this reason, this property is also sometimes referred to as the confinement property.

Simple property. The simple property specifies that a principal must have a security label more privileged than the labels on the objects they may read.

Tranquillity property. Finally, the tranquillity property requires that when a system is currently accessing an object, the security label of the principal and object involved must not be changed.

The tranquillity property demonstrates that this model also needs to maintain information about the current state of the access control system to guarantee correct behaviour. We do not provide details here, other than to note that a finite state machine can be constructed to ensure that label modifications are performed at safe times. Such a finite state machine will obey the conditions of the Basic Security Theorem (BST), which describes how, given initially 'secure' states, sequences of valid requests will generate other 'secure' states.

McLean extended the Bell-LaPadula Model (see [McL85, McL90]) after constructing an example that shows that the separation of the BST from practical notions of security leads to flawed behaviour. The particular types of requests made in this example were valid with respect to the BST, but changed security labels to allow all subjects to read all objects – hardly secure in an intuitive sense.

Biba model

The Biba model [Bib77] is closely related to the Bell-LaPadula model, however it is considerably more general. It broadens Bell-LaPadula in two main areas:

Dynamic integrity levels. The Bell-LaPadula model assumes static compliance to its lattice of security labels. The Biba model has two modes of operation. The first is identical to Bell-LaPadula. The second relaxes the constraint of static security classifications, providing the ability for subjects and objects to be reclassified based on their contact with subjects and objects at other security levels. The greatest lower bounds, and least upper bounds for the pairs of objects in the security lattice are used for this reclassification.

Invocation notions. As mentioned above, Bell-LaPadula focuses on the reading and writing of data. The Biba model adds the notion of software invocation to the set of access modes. This facilitates management of security properties in cases where software objects, possibly residing at different security levels, perform jobs on behalf of a principal.

These extensions increase the Biba model's applicability to the functions of computer operating systems, but still leave a military-style inflexibility. Clark and Wilson [CW87] proposed that there are numerous security requirements relevant to commercial environments which focus on integrity rather than focusing entirely on disclosure.

Chinese Wall model

Similar in motivation to Clark and Wilson [CW87], the Chinese Wall model [BN89] was stimulated by business operations where client databases need to be kept separate from each other. As with other MAC schemes, this model is concerned that information does not flow between certain classifications. In contrast to the other models we have presented, the Chinese Wall model does not place the different classifications into a partial ordering.

A list of company datasets is stored, and connected with each dataset in that list is a list of competitor datasets that should have enforced separation. The intuition is that until company datasets are 'sanitised' into a dataset containing no critical information, this dataset should not be accessible to subjects who also work with their competitors.

To ensure information flow constraints are maintained, a matrix with boolean values records whether each subject (on one axis) has ever interacted with each object (on the other axis).

Two properties then ensure valid access control:

ss-property. This property states that a subject will only be allowed access to an object if two conditions hold with respect to the other objects they have accessed. Each of the other objects they have accessed has to either be in the same company dataset, or in a different company dataset, but one not in competition with the company connected to the subject's object request.

***-property.** This property states that a subject will only be allowed write access to an object if they do not have read access to any other company's objects where that company has non-sanitised data. A company with non-sanitised data will have a non-empty competitor dataset list. If the *-property did not hold, subjects could write confidential data to another company's object, which after transitive closure may include one of their competitors.

An important contrast between the Chinese Wall and Bell-LaPadula models is that the former requires access rights checks to be performed during all state transitions, whereas the latter has static access rights. That is, with the Chinese Wall, users' rights are potentially modified by all interactions with the security system (e.g. reading previously untouched objects).

2.2.2 Discretionary Access Control (DAC)

For most purposes, MAC schemes are too restrictive to be convenient. In particular, exceptions to general rules are very hard to manage. Most common operating systems thus instead provide Discretionary Access Control (DAC); users' rights can be managed in an *ad hoc* fashion. Of course it is at the administrator's discretion to create structures which mirror MAC schemes, but it is unlikely that the underlying security system will provide much support to enable a check for consistent security state.

In this section we examine a number of common DAC schemes and principles. In the section following this, we discuss the Role-Based Access Control class of DAC schemes.

The access control matrix

Lampson [Lam71] first presented a model involving an access control matrix. Although a very simple idea, it has provided the basis for many other access control approaches. In his model, security state involves the tuple (S, O, M), where S is a set of subjects, O is the set of objects (which also includes S as a subset), and M is the access control matrix. Every subject in S has a row in M, and every object in O has a column. Each matrix cell M_{so} provides a given subset of A, which is the set of all possible access modes. Predictably, cell M_{so} indicates the access rights the subject s has over object o.

In large-scale systems, the matrix M is likely to be sparse. This is simply because increasing the number of entries in M leads to more complex, and less verifiable or intuitive security policies. As a consequence, M is unlikely to be stored explicitly (even less so for higher-dimensional models of M such as [Kno00]). Instead, usually either columns

Principal	**Rights for each object**	
	Accounts database	Colour laser printer
Alice (accounts)	read, write	
Bob (marketing)	read	print
Charlie (IT)		print, delete job

Table 2.2: An example access control matrix

or rows of the matrix M are stored, leading to access control lists and capability-based representations respectively. These are described in the next two sections.

Access Control Lists (ACLs)

Each column of an access control matrix represents an Access Control List (ACL). ACLs are particularly popular for operating systems security on file-systems, since it is convenient to store ACLs (or references to repeated ACLs) along with each file (in this case the security object). Thus each file describes which subjects have given rights over it.

Of course the ACL-object coupling may also be a negative feature, since it does not directly assist discovering the total set of objects that a particular subject has permission to access.

Capability-based systems

Rather than storing columns of an Access Control Matrix, capability-based systems instead store rows (see [Lev84]). Thus they encode, for each subject, which objects are accessible. Predictably, such a system has the inverse problem of ACLs: it is difficult to assess quickly all the subjects who can access a particular object.

Whilst we broadly define capability systems as focusing on storing rows of access control matrices, generally they are able to split each row into smaller parts. Thus capabilities are able to be disconnected from the subjects to which they belong. Whilst such delegation schemes may cause difficulty in proving the safety of a given access control environment, they also greatly increase the flexibility of system operation. Most non-computer domains define lines of delegation such that if a principal is unavailable, the next senior position can temporarily take over responsibility. Capability-based delegation mirrors this intuitive organisational practice.

This thesis examines the development of an active, distributed capability-based access control system.

Harrison-Ruzzo-Ullman model

A good example of a DAC framework is the Harrison-Ruzzo-Ullman (HRU) model [HRU76]. As in the Lampson Model, the HRU model defines an access matrix M, a set of subjects S, a set of objects O and a set of access rights A.

The main feature of the HRU model is that, rather than assuming M is static, it explicitly investigates how M may change during system operation, and whether safety properties continue to be maintained.

To describe dynamic behaviour, the HRU model specifies how an application can define its 'commands'. Each command takes a particular subset of S and O as arguments, and has two distinct sections. The first section does a number of conditional checks based on whether certain rights (r) are in cells of M corresponding to the command's arguments. If this conditional section evaluates to a true value, a series of operations is performed. Each operation can be in one of six forms:

- Enter right r into M_{so}
- Delete right r from M_{so}
- Create subject s
- Delete subject s
- Create object o
- Delete object o

To examine safety, the HRU model defines the concept of a security 'leak'. A state of the access control system is said to leak a right r if that system executes a command which generates a new matrix M'_{so}, and some cell M'_{xy} now contains a right which was not in cell M_{xy}. An access control matrix M is said to be 'safe' with respect to a right r if no sequence of commands on M can cause it to leak r.

Unfortunately the safety result can be shown to be undecidable in general. One approach is to limit each operation to perform only a single command. Using such 'mono-operational commands' the safety of M with respect to r is decidable. This is a strong restriction however. More usefully, it has been shown that if the number of subjects is finite, then the behaviour of arbitrarily complex authorisation systems is decidable.

2.2.3 Role-Based Access Control (RBAC)

Most of the security models presented above are too formal for use in common operating systems. Of course there are classes of operating system which have explicitly verifiable security models (c.f. the US Military Orange Book security classifications), but their restrictiveness tends to mean these are generally only appropriate for security-critical use.

Within DAC systems, it became increasingly clear that maintenance of ACLs or capabilities (or indeed an Access Control Matrix directly) presents administrative problems.

Many security administration features involve managing sets of privileges or principals; security publications thus began to suggest the introduction of another level of abstraction – the 'role'.

The reasoning behind this concept is that subjects or principals map to 'roles' rather than directly to privileges. Roles can then have sets of privileges attached to them. Thus

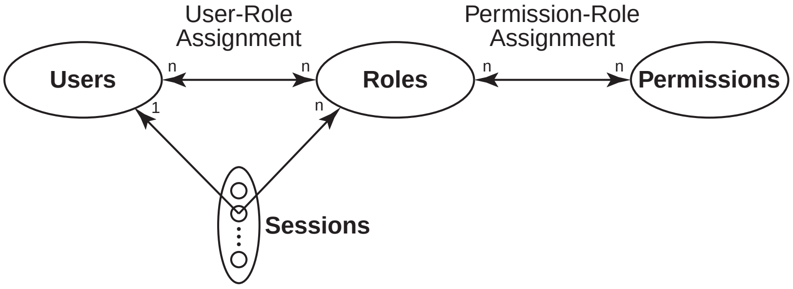


Figure 2.2: $RBAC_0$ example

all similar subjects can have their accessible privileges modified simultaneously. Roles may also increase the safety of updating the privileges of a subject when they change their work with respect to the security system. In the literature, roles are often attributed to a 1988 paper by Lochowsky and Woo [LW88], although the basic idea had probably been considered earlier. One of the main contributers to RBAC research has been Ravi Sandhu [SS94, San95, San96, SCFY96].

Although there has been a large amount of research into Role-Based Access Control (and, indeed, entire symposia devoted to the subject), complete RBAC models are still surprisingly sparse in real-world applications. Most uses of the term 'role' in application configuration are only in the most basic sense of the linking of subject to privilege or object mappings (e.g. Microsoft Windows security, or J2EE [Sha01] systems such as JBoss and web-serving infrastructures such as JetSpeed or Apache's TomCat).

Before exploring how the term 'role' can be interpreted in more detail, we ground our discussion by presenting an overview of the NIST RBAC standards.

NIST RBAC standards

The American National Institute of Science and Technology (NIST), has specified a set of four standard models for Role-Based Access Control (see [FK92, SFK00]). Recently the NIST standard has been adopted as an American National Standards Institute (ANSI) standard (ANSI INCITS 359-2004). These standards define models known as $RBAC_0$ through to $RBAC_3$. We feel that the actual structure of this four model framework is of questionable justification, however the first three models define important RBAC features and are a good starting point for discussion:

$RBAC_0$**.** This is the most basic form of RBAC (see figure 2.2). Roles merely link principals to privileges. Mathematically, given a set of subjects S, a set of roles R and a set of privileges P, there are two mappings of interest, the first of which is the 'user-role' assignment function. Given $s \in S$, this function will return the roles in which this subject is a member – a subset of R. The second function is the 'permission-role' assignment function. Predictably this function, given $p \in P$, indicates the subset of R which has been granted this permission.

$RBAC_1$. This level of RBAC subsumes $RBAC_0$, and introduces role-hierarchies. Instead of users mapping through roles to privileges, now a role may itself map to another role. The more senior role, through its junior role, has access to all the privileges of both roles. Note that a partial ordering must be defined among the roles for such a concept to make sense – were role A to map to role B, in addition to role B mapping to role A, roles A and B become equivalent (and should not exist as separate roles at all).

$RBAC_2$. The ordering of classification is deceptive here – in fact $RBAC_2$ subsumes $RBAC_0$ (but not $RBAC_1$), and adds the concept of 'constraints' for state transitions. In other words, a user's authority to enter a role may depend on more conditions than merely a requirement for an entry in the user-role (or transitively through role-role) assignment functions. This is a significant extension from basic RBAC, that strictly requires user sessions since otherwise it is unclear when the constraints should be tested. When roles have to be explicitly activated within the context of a session, constraints can be tested at activation time. Many systems ensure that sessions cannot be infinitely long, thus determining the minimum frequency for constraint checking. A number of RBAC architectures also support explicit revocation of roles. The OASIS architecture [BMY02a] allows users to specify in each policy rule whether or not certain conditions are checked in an ongoing manner, versus just at role-activation time.

$RBAC_3$. This level of RBAC combines the extensions of $RBAC_1$ and $RBAC_2$. Given these respective extensions are orthogonal, a better labelling approach could have avoided the need to define this level in its own right.

What is a role?

So far, our treatment of what the term 'role' means has been largely technical. Its abstraction, versus direct user-privilege mapping, is administratively appealing, but on what basis should roles be defined within a large-scale organisation? Deriving sensible role definitions is usually referred to as *role engineering* (see [SA03, KSS03, NS02, KKSM02, BLM01, SMJ01]).

In answering the question about an employee in a business, "what is John's role in the organisation?", it is likely that the most direct answer will define their position within the business employment structure. For example, John may be a sales manager. Role hierarchies generally assume that organisational hierarchies can be at least partially mapped into RBAC security specifications.

It is also possible to define roles in a much more task-oriented manner, however – often referred to as 'functional roles'. In this case, John may have many roles, one of which might be to "distribute internal sales reports", for example. Such task-based roles make more sense from the perspective of 'sessions'. If the above role permits John to temporarily examine and select a list of potential report recipients, it is highly unlikely he will be doing so for a high proportion of his working hours. On the other hand, it seems strange to have to explicitly activate a role of "sales manager" within a user session when this is a

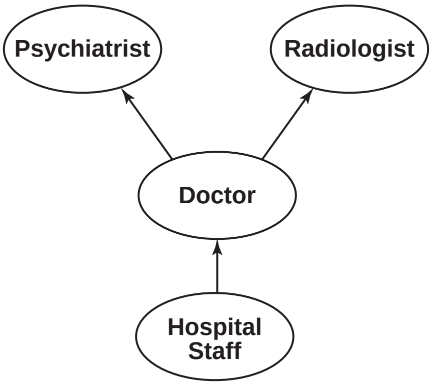


Figure 2.3: A role hierarchy

persistent credential.

Role hierarchies and delegation

If roles are to mirror organisational structure, it is appealing to map any orderings of seniority within the organisational structure to the role structure. This is the basis of role hierarchies, as included in $RBAC_1$ and $RBAC_3$ (§2.2.3).

Generally, role hierarchies involve the more senior roles being able to gain all the rights of the roles they dominate within the hierarchy. Given there are often parallel areas of work within an organisation, role hierarchies are usually represented as a lattice (see §2.2.1). This means that whilst managers of any department will dominate the roles of their employees, there is no defined ordering between the managers – they cannot be active in each others' roles.

The similarities between the lattices involved in role hierarchies and those used in MAC security labelling have led researchers to examine MAC-style security properties that can be enforced within role hierarchies [NO99, Osb02]. For example, the concept of a least sufficient 'maximum' role for a given set of privileges comes from the least upper-bound structure within the underlying lattice (see [Cra03] for further such analyses).

Whilst role hierarchies can facilitate privileges travelling upward to more senior principals, there may well be a need for temporary delegation of senior privilege down to more junior principals too (such a mechanism is proposed in [NC00]). If a senior principal is temporarily unavailable, progress in an organisation may depend on the ability for a junior principal to act above their normal security level. Falling outside the normal privilege ordering, such delegation can often be viewed as providing a particular capability to the more junior principal. Researchers have suggested models (e.g. [ZOS03]) which limit the ability for that principal, then, to further delegate privileges in order to better track the use of these temporary capabilities.

At first sight it may appear that privilege delegation is significantly more risky than the ordering imposed by a role hierarchy. However, we believe that it is actually the role hierarchies themselves that are dangerous, in that they provide a false sense of security.

In fact most useful forms of privilege ordering in organisations do not follow such a strict hierarchy. It would seldom be the case that the Chief Executive Officer of a large organisation would have sufficient understanding to perform the work of a staff engineer. Indeed it would probably be explicitly unsafe to let the CEO have access to these privileges.

Thus responsibility and function do not need to be merged. High-level management should be responsible for the actions of their employees, and should be able to authorise certain actions, but this is not equivalent to high-level management actually having direct access to the privileges they authorise. An example is where there is an explicit separation of qualification, say between a triage nurse in an accident and emergency department and the doctor on duty there. In this case, the nurse determines the order in which the patients who appear will be treated. However, whilst the nurse authorises the actions of the doctor on duty, they should not be able to use the doctors' privileges themselves.

The OASIS model includes a notion of 'appointment', through which principals can bestow credentials upon other principals so they may activate certain roles. Appointment is more general than delegation, but has the advantage that each relationship must be made explicit in the security policy.

Groups

Another common extension to basic RBAC is the inclusion of group or team support. Group support in RBAC simply acknowledges that the grouping action provided by roles tends to be more privilege-focused. It is thus possible to add another grouping construct to the principals themselves, and sometimes connect roles with these principal groups. Team-Based Access Control [Tho97], and Coalition-Based Access Control [CTWS02] are examples of RBAC models that include group extensions. The former focuses on groups within one organisation, whereas the latter covers groups which may span different organisations and change membership frequently.

Group support can intuitively handle certain types of cardinality constraints. For example, whilst the members of a team may all have access to a particular role, it may not be acceptable that they all activate it. Instead RBAC team support can specify, for instance, that only one member at a time of a particular team can activate this role.

Parameters

If role engineering is done on the basis of an organisational structure, it is important to point out that the size of an organisation does not necessarily relate to the number of distinct roles the organisation will require. Many businesses will have detailed organisational structures that provide a rich basis for role definition in the access control system. However, in a university, for example, there are large numbers of principals (e.g. students), who have very similar role requirements.

In such a situation, rather than create a vast array of specific roles for these students, a much more scalable approach is to parameterise the roles themselves. So in a parameterised RBAC system, a role 'student', might be tagged with attribute 'studentID'. Thus certain policy rules can be specified which allow privileges to students irrespective of which specific student they are.

Parameterisation will generally increase the potential complexity of policy evaluation, however. This is because the policy languages will be augmented with syntax to specify constraints based on particular role attributes. For example, the properties of a 'studentID' attribute may allow the policy system to distinguish between part-time and full-time students, and allow different privileges for the two groups. Such rules also make it harder to determine the overall behaviour of the policy system, since fine-grained attribute interactions may need to be taken into account.

Environmental interaction

RBAC systems that include constraints often have a need to interact with the environment outside the access control system itself. Otherwise decisions could only be made on the basis of presented credentials alone.

One common example of an environmental interaction is a dependence on time – e.g. it may only be acceptable that shift-work employees activate certain roles within specific time ranges. Another common environmental interaction is with external databases systems. This may be useful when checking whether a particular credential is within an externally-managed list. Any useful policy language will allow administrators to add specific credential checks into the policy definitions, however this will not provide a very flexible representation. Storing information in a nearby database has the advantages that the database will focus on indexing sets of data, and can be manipulated by existing interfaces.

The two environmental interactions discussed above involve performing external checks when attempting to satisfy policy constraints. Environmental interaction may also be useful for translation purposes. For example, in a parameterised RBAC system, the digitally-signed persistent credential of a principal may contain an 'ID' attribute with global significance – say a doctor identifier issued by the General Medical Council (GMC) in the UK National Health Service (NHS). However, the active roles at some particular hospital may be more useful if in the form of a local identifier. In such a case a local hospital database would contain information connecting a hospital 'ID' attribute with the wider-scope GMC 'ID' attribute. Policy rules could then interact with this local database, even though it is outside the access control system itself, to translate the persistent credentials into active roles of local significance.

Of course such environmental interactions pose risks in terms of information flowing out of the access control system. Applications need to be designed with careful consideration of where (possibly unintentional) information flows may compromise security. We introduce the notion of *policy contexts* which help monitor such policy administration concerns in section 4.4.2.

Self-administration

Theoretical models of access control systems are often more concerned with how security properties can be proved for a given system state than how to administer real-world applications. In real applications, a major consideration has to be policy administration. It will seldom be the case that the first deployment of a policy system will contain all necessary policy rules, nor will these rules remain unchanged during system operation.

Apart from considering the impacts on security state given changing access control policy (e.g. what to do with currently active roles if the rules which originally caused them to be activated are modified), actual changes to the access control policy should be controlled.

The concept of access control systems controlling access to their own policy has been widely explored. Commonly-cited examples include Sandhu's Administrative RBAC models (such as ARBAC97 [SBM99]). Essentially, certain privileges are chosen to be interpreted as commands to change the access control state itself. A comprehensive set of administrative privileges for policy modification is presented in [Bel04].

A particularly appealing property of allowing the access control system to guard its own policy is the ability to define a hierarchy of administrators. A top-level administrator may limit the areas in which the administrators below them can modify access control policy. In large-scale distributed access control systems this process is likely to be the only sensible manner in which policy can be managed, otherwise the work of central policy administration becomes too onerous.

Meta-policy

Self-administering policy is one example of what some authors refer to as 'meta-policy': namely, policy which does not directly relate to providing privileges over protected resources to principals, but instead over the policy itself. Self-administration is simply another type of policy, although we acknowledge that it is at a higher level than the policy directly applied for protection of resources.

Other types of higher-level policy have also been suggested (see [BM02a]), two examples of which are 'compliance' and 'interface' policy. In the former, high-level policy administrators aim to enforce particular structures of lower-level policy specifications without excluding local administrator freedom. For example, a high-level NHS policy might be that, in emergency situations, doctors can acquire a certain set of exceptional privileges. Belokosztolszki describes this high-level policy as a 'meta-policy'. Any given hospital's policy administrator can define their own local policy rules, provided that they can prove compliance with high-level policies.

'Interface policy' relates to how different organisations communicate with each other. Again in the health domain, it may be desirable that doctors who normally practise at one hospital are able to acquire privileges at another hospital. If the hospitals define different types of 'doctor' credentials, interface policy will be necessary to determine how to translate credentials from one hospital to credentials suitable for activating roles at the other.

Policy languages

Any discretionary access control system must have a human-configurable representation for the rules it enforces. In the case of $RBAC_0$, the policy 'language' is really just the user-role and privilege-role relationships. More complex policy languages are necessary in access control systems that support constraints.

The most basic step up from user-role and role-role relationships are access control systems where principals must possess a set of role credentials before they are permitted to activate some target role. The next iteration of policy language complexity includes environmental constraints such as the database and time-of-day checks introduced in section 2.2.3.

Policy languages become more complex again when they include parameterised RBAC elements. The OASIS and EDSAC languages are in this class of complexity. At this level it is important to guarantee termination of policy rule evaluation. In the OASIS language, this is achieved by showing that the parameter binding in a policy rule is topologically sortable, to obviate any cyclical parameter dependencies. Further, OASIS rules cannot be recursive, thus there is no risk of infinite execution loops.

At the other extreme of expressiveness are conventional programming languages. Procedural languages such as Java can be used to implement policy-checking code, but the policy representation quickly becomes implicit and obscure.

A compromise is offered by a language such as Cassandra [BS04a], recently developed at the University of Cambridge to provide an expressive RBAC policy language. Cassandra is a datalog-based language, which defines different constraint sets, so policy administrators can trade-off code complexity against reduced expressiveness.

Another such language that balances expressiveness with language complexity is Ponder [DDLS02, DDLS01, Dam02]. Ponder is an object-oriented language with a large feature set including obligation policies and negative permissions.

2.2.4 Role-aware security

Whilst the NIST RBAC standards provide a good grounding for RBAC discussions, they are severely limited in their applicability to modern access control systems. In particular the models proposed do little to support dynamic behaviour or the interaction between access control agents in a distributed system.

Recent research has generally turned to more comprehensive models for privilege management. As a consequence, tracing the route back to the NIST standards becomes increasingly difficult. Take for example the X-GTRBAC administrative model [BJBG04]. Firstly this is a model relating to the administration of another model: X-GTRBAC [Bha03]. Dissecting that acronym, we first remove the 'X' which indicates an implementation based on XML [W3C00]. Thus X-GTRBAC should be viewed as an implementation of the GTRBAC model.

The 'G' and 'T' of GTRBAC represent 'generalised' and 'temporal' respectively. The temporal extensions to RBAC [BBF01] were proposed first and present a policy language to allow time constraints over RBAC behaviour, in particular TRBAC distinguishes between 'role enabling' and 'role activation'. This parallels an intuition built into OASIS research for a long time, namely, that roles should be active within some notion of a session, to allow for prerequisite predicate checks to be performed at the time of activation and possibly during the session's duration.

The generalisation of TRBAC [JBLG01, JBG02, JBSG03] involves adding support for role deactivation as well as role activation, in addition to managing the periodic time intervals likely to occur in real organisations (e.g. authorisations which depend on particular hours in the day, days in the week, and/or days in the month).

Many researchers have turned away from attempts to connect intimately to the NIST RBAC standards, but without discarding the usefulness of the fundamental role abstraction, and the past research surrounding it. Sandhu himself has focused increasingly in recent times on the Usage Control (UCON) model (introduced in [PS02] and extended in [ZPPPS04]), since it relates to ongoing privilege management in a manner NIST RBAC is unable to support. Sandhu still continues to propose RBAC extensions too [AS03, ZOS03].

Ao and Minsky present the case for role-awareness in [AM04]. Next we provide an overview of a number of important research projects providing policy specification languages that use roles, but do not rely unduly on NIST RBAC.

Cassandra

The Cassandra language [BS04a] is good example of a role-aware policy language. As mentioned above, it uses an extended form of datalog. Datalog itself is like Prolog (for an introduction see [CM94]) with the strict limitations that terms cannot have functional form. Also, control-flow mechanisms such as backtracking do not exist, there are no built-in predicates, and there is no way to include negative literals in the body of a predicate. This is too limited for expressing most policies (e.g. [BS04b]), so Cassandra augments each datalog clause with constraints from a given constraint domain.

Becker has defined a number of constraint domains, each having increasing expressiveness, but predictably coupled with increasing complexity for evaluation. In all cases they provide proven bounds on the time until termination, which is an essential safety property when computing security policy predicates. The two examples given are $\mathcal{C}_{min}$ and $\mathcal{C}_0$. The former's predicates can only take the form of boolean values, conjunction and disjunction of predicates and equality all over simple terms. The latter can additionally handle inequality, and subset checks over a vastly extended variety of entities including tuples, sets, projections over tuples, functions, unions and intersections.

The main security-specific aspect of Cassandra involves its role awareness; a number of specific predicates marshal role activation and deactivation. canActivate and canDeactivate are guards for the assertion and removal of hasActivated and isDeactivated clauses, each of which specify a principal and a role. Further, Cassandra includes the canReqCred predicate – if true, a given principal can request a credential from a remote service.

Being a very general policy language, it supports the specification of complex policy including role hierarchies, separation of duties, appointment, and various types of revocation.

At the time of writing, this research has been language-focused, although we hope in future to build Cassandra implementations using the messaging and credential representation infrastructure of EDSAC.

Law Governed Interaction (LGI) and Rei

The concepts behind Minsky's Law Governed Interaction (LGI) were first explored in [Min91], although this work was found to be too general to be practically implemented. LGI limits the manner in which 'laws' can be described, and is presented in [MU00]. Even so, LGI is still considerably more general than most access control systems, describing a coordination protocol between distributed agents rather than a system specifically oriented to security.

Agents in a community choose which laws they will adopt (generally encoded in a Prolog-like language), and operate in accordance to them. An element of trust is required in that agents enact valid actions according to the laws they uphold, and thus in a distributed manner. Of course another agent with access to the same information, and adopting the same laws, can check the first agent is not cheating, although this verification is not mandated by the LGI approach.

Similar in high generality, the Rei language [Kag02] aims to model fundamental deontic notions of rights, prohibitions and obligations. In addition, Rei has the notion of a dispensation, namely, cancellation of a previous obligation that is no longer relevant to a principal. Rei also uses the Prolog language for its implementation, and defines a Resource Description Framework (RDF) [W3C99] ontology for its policies. The Rei policy interface is facilitated through a comprehensive set of Prolog clauses (optionally accessible via its Java wrapper).

In summary, these interfaces allow the definition of policy constraints, the possible actions in a given policy system, policy objects (rights, prohibitions, obligations, dispensations), speech acts (mainly for supporting delegation), and meta-policies (policies controlling policy interpretation – namely the management of conflicts). Unlike LGI, Rei's meta-policy interface is explicit, and serves the function of either ordering clauses, or ordering the modalities within clauses of the same precedence.

In terms of its relationship to RBAC, a notion of roles can be encoded into the laws adopted by a community, or be used in the actions and objects of REI, and thus both models can enforce types of role-based access control.

Ponder

The Ponder policy specification system [DDLS02, DDLS01, Dam02], is an object-oriented, declarative policy language for controlling policy-based networking (PBN) equipment or software written in the Java language. It is particularly suited to engineering network-wide policy over individual policy-aware network control devices, since it provides a common language abstraction over the heterogeneous components in such a security network.

The Ponder authorisation policy syntax is:

```
inst ( auth+ | auth- ) policyName "{"
   subject [<type>] domain-Scope-Expression ;
   target  [<type>] domain-Scope-Expression ;
   action           action-list ;
 [ when             constraint-Expression ; ] "}"
```

The auth+ and auth- variants embody Ponder's support for positive and negative authorisations. Negative authorisations are intuitive, but do complicate the analysis of policy. Since Ponder policy specifications are compiled into configurations for the various underlying devices, it is realistic to require that all potentially conflicting elements of policy be centralised for static analysis. Ponder also provides group support, roles and important real-world management requirements such as controlled delegation of rights.

As a powerful extension to conventional RBAC, Ponder's roles can have attached obligation policies. The syntax for such policies is:

```
inst oblig policyName "{"
   on               event-specification ;
   subject [<type>] domain-Scope-Expression ;
 [ target  [<type>] domain-Scope-Expression ; ]
   do               obligation-action-list ;
 [ catch            exception-specification ; ]
 [ when             constraint-Expression ; ] "}"
```

However, since Ponder approaches the policy management problem from the language downward, device configuration specifics need to exist in separate modules. This has meant other researchers have worked instead from the network hardware upward, which has resulted in the Common Open Policy Service (COPS) protocol [DBC^{+}00], and the access control relevant portions of the Policy Core Information Model (PCIM) [MESW01]. Ideally these two approaches could merge, but there is such a huge amount of heterogeneity in any real implementation that such integration is likely to be a massive undertaking – particularly given that new technologies are continuing to be released. Indeed technologies such as RADIUS [RWRS00] are designed merely to bolt together other technologies (authentication technologies and authentication record management in this case), but even so are remarkably complex themselves.

The XML Access Control Markup Language (XACML)

The XML Access Control Markup Language (XACML) [OAS02] is a technology developed by the Organisation for the Advancement of Structured Information Standards (OASIS – hence there is some potential confusion with our usual use of the term). It provides a language for specifying access control policy, the structure of XML messages that request access to resources, and the structure of the messages responding to these requests.

XACML is a highly modular language, which explicitly supports a variety of different forms of combining logic when reaching a decision on a given request. For example, the inbuilt Deny Overrides combining algorithm will cause a negative result if any rule returns a non-positive result. XACML includes well-defined points where its existing data types, functions and forms of combining logic may be extended.

In order to facilitate numerous different types of access control, XACML Attributes unify notions of identity, functional role, position within an organisation, and so forth. These Attributes may be converted into intermediate form by Functions within a Policy Decision Point (PDP) in the course of rule computation. It also provides a standard mechanism for these Attributes to be digitally signed by some other authority.

When provided with a resource request, and the user's attributes, a PDP searches for the rules which match the user and the target object, determines these rules' truth values, and then applies its configured combining logic to ascertain the final response. Any given request will either be permitted, denied, be declared indeterminate (e.g. attributes were missing so a decision could not be made), or given a 'not applicable' response (i.e. this PDP is not the place to make the user's request).

Rules (which are built up of Conditions) are contained within Policy XML elements. In turn, PolicySet elements may contain other PolicySets, Policy elements, or references to remote policy data. Thus there is no need to centralise policy storage at the PDP(s) which will ultimately use it.

XACML stores a simplified version of the Subject, Action and Resource into an entity called a Target which is used to increase the speed of finding policies which pertain to a particular request.

Security Assertion Markup Language (SAML)

Another important technology for services interacting over the Internet is the Security Assertion Markup Language (SAML) [HM04]. Like XACML, it is also a standard proposed by the OASIS consortium. The primary goal of SAML 1.0 was to manage authentication so as to to facilitate single sign-on (SSO) capabilities for the World Wide Web – i.e. allowing an identity to be portable across enterprise boundaries on-line. At the time of writing, SAML 2.0 is being developed, and includes more features related to web-services, such as trust negotiation between SAML servers, request encryption and various discovery protocols.

The SAML language describes two main components; the set of assertions, and the request/response protocols which will use these assertions. A SAML authority can make one of three types of assertion; authentication, attribute and authorisation decisions. For SSO, authentication is the key assertion, where a SAML authority indicates that a specific subject was authenticated by some particular means at a given time. Attribute assertions link attributes to particular principals, and authorisation assertions record the outcome of a PDP decision.

The protocols that SAML defines allow providers to request one or more assertions, to request that a principal be authenticated, and to request retrieval of name identifier mappings. Other protocols include those to manage the registration of names and the distributed logging out of sessions.

SAML has been highly successful as it provides a unifying means to solve problems beginning to face an increasing number of web-hosting authorities.

WS-Policy versus the Web Services Policy Language (WSPL)

Web Services are an important emerging technology which are currently lacking in policy control. The OASIS and EDSAC architectures have, to date, modelled policy for the sake of controlling the actions of humans agents within a system. Web Services policy, on the other hand, focuses much more on the automatic negotiation of interactions between machines (which will be an important aspect of building the 'Semantic Web' [BL98]). Thus, in addition to authorisation policies, negotiations will include Quality of Service (QoS), privacy, message delivery semantics, and similarly low-level system concerns. Many of these aspects of policy are required because Web Services trade on the open Internet, as opposed to working within a bounded security domain.

Whilst both WS-Policy and WSPL (the Web Services Policy Language) are aiming to achieve the same goal, at the time of writing there is a standards conflict between them. WSPL is using an open standards organisation (and is based on XACML), whereas WS-Policy is being developed by a private consortium. Anderson provides a comparison of the two in [And04], although she clearly supports WSPL.

2.2.5 Distributed RBAC infrastructures

We conclude our overview of access control systems by looking at two particular research projects that focus on infrastructure and protocols to support distributed RBAC. Our EDSAC design is intended to capture the best features of both.

The PERMIS access control architecture

The PrivilEge and Role Management Infrastructure Standards validation (PERMIS – see [CO02a, CM03, CO02b]) distributed access control architecture infrastructure has a strong reliance on the ISO/IEC 10181-3 standard (the access control framework section of the OSI security framework). See [BMCO03] for a comparison of the key features in PERMIS and OASIS.

The particular focus of PERMIS is a separation between the digital credentials used for authentication versus those used for authorisation. The standard PKI (public key infrastructure) approach to identity is proposed, whereby a particular X.509 certificate corresponds to the identity of a principal. Protection of this PKI certificate is managed through a private key password, or other authentication means such as biometrics. It is expected that these PKI certificates will be issued by a high-level certificate authority (e.g. a government department), and will normally never need to change or be re-issued.

The actual attributes on which access control decisions are based, however, are recorded in a different type of X.509 certificate. As defined in ISO/IEC 10181-3, these are Privilege Management Interface (PMI) certificates. These credentials are linked to the identity of a principal through the public keys of the X.509 identity certificates. Separating the "permanent feature" of identity from "changeable attributes" such as authorisation to operate in some organisational role has the advantage that the certification authorities may also be very different in scale. Any given university could chose to sign X.509 PMI certificates which link to the more global identities of their students' PKI certificates.

To manage these collections of X.509 certificates, PERMIS uses the Lightweight Directory Access Protocol (LDAP). LDAP directories are designed to handle X.500 data effectively, although as described in [Cha03], there are certain problems reconciling datatypes in LDAP versus the ASN.1 X.509 certificate encodings.

PERMIS further uses X.509 certificates for its policy representation. These policy certificates indicate the prerequisites to acquire some given privilege. The advantage of using certificates for policy is that they, too, can be stored in the existing LDAP directories, and this allows different organisations to define their own policy fragments, avoiding the need for centralised policy maintenance.

Distributed RBAC for dynamic coalitions (dRBAC)

The Distributed Role-Based Access Control (dRBAC) [FPP+02, FPP+01] project is also based on X.509 certificate infrastructure. Like PERMIS, it recognises the need to support credentials from external trust domains, and that PKI can be used to achieve this aim. One of its main features is the ability to handle dynamic coalition environments, where trust relationships are easy to set up, but provide varying levels of authorisation based on the resource owner's level of established trust in the principal. Various types of delegation are supported in dRBAC, including a notion similar to OASIS's appointments – subjects from outside a given domain may delegate roles to subjects within that domain if they are trusted to the appropriate degree.

The dRBAC architecture also manages credential discovery. 'Wallets' are locations on the Internet which contain user credentials, or pointers to credentials contained in other remote wallets. All Subjects, Objects and Issuers structures include a 'discovery tag', which specifies their authoritative home location, liveness monitoring requirements, and the contexts and roles in which this principal can legally operate.

Finally, dRBAC recognises the need to integrate credential monitoring infrastructure into the access control system itself – this has always been recognised in OASIS. Like the EDSAC architecture, dRBAC uses publish/subscribe messaging (see §2.3.3) to update credential state.

2.3 Scalable message delivery systems

This thesis proposes building an active access control infrastructure over distributed systems using content-based message delivery. Whilst we have been involved with some research into combining security with content-based message delivery systems [BEP+03], we predominantly remain a consumer of existing message delivery systems. This section describes the type of message delivery system we have used, and why it is suitable for applications such as ours, that require a high degree of scalability.

Message delivery systems perform the computing equivalent of a postal service; senders are able to cause receivers to gain access to messages they have transmitted.

2.3.1 Internet message protocols

The Open Systems Interconnection (OSI) protocol stack defines seven layers at which Internet protocols operate, from the physical protocol (e.g. 802.11g wireless or Ethernet; 802.3 CSMA/CD) up to the application level. We are generally interested in protocols closer to the top of this stack, since they abstract over the various choices for lower level implementations.

Most Internet applications use the Transport Control Protocol (TCP/IP) and/or the User Datagram Protocol (UDP/IP) to transmit data. Whilst other choices such as Novell's IPX protocol and Apple's Appletalk are still in use too, they are increasingly providing the capacity either to operate similar services over TCP/IP, or at least to tunnel their alternative protocols over TCP/IP. This is primarily so that they can take advantage of the cheap and plentiful hardware and software components that have adopted TCP/IP.

The Transport Control Protocol (TCP) and User Datagram Protocol (UDP) differ in that the former provides a connection-oriented stream between a sender and a receiver. The latter indicates sender and receiver, but does not indicate that any particular stream ordering should be observed at a receiver. TCP is a much more complex protocol, since it must manage retransmission of lost packets in a stream, as well as considering effects such as network congestion and rate control.

Both protocols rely on the Internet Protocol (IP) to specify the sender and receiver of their data packet or stream. The common IP version 4 (IPv4) [Pos81] provides for 32-bit addresses, however it divides the address space between network and host addresses. A network mask indicates how many of the bits select the network, the remainder determine the host within that network.

Due to problems managing the hierarchical nature of IPv4 address allocations, there is currently a scalability problem – as more and more devices require IP addresses, the world is likely to run out of them within the next few decades. Various solutions exist, including using Network Address Translation (NAT) to allow multiple machines to share a single address. A better long-term solution is to upgrade to the Internet Protocol version 6 (IPv6) [DH98], which expands the raw address space to 128-bits.

Hierarchical routing of data packets through the Internet works acceptably well for communications between specific hosts. However, many applications benefit from allowing many-to-many communication. The Internet protocols include support for multicasting – the sending of a single message to multiple recipients simultaneously – however current network hardware does not universally support multicasting, and there are many research problems outstanding in managing multicast groups. Thus for the moment multicast remains useful predominantly at the local-area-network (LAN) level.

2.3.2 Content-based message delivery

The way in which IPv6 expands the address space attends to scalability issues relating to the architecture of the Internet, but does not solve problems with respect to the scalability of its activities. Whilst all hosts on the Internet may be locatable through a standard addressing scheme, this homogeneity masks the radically different levels of demand placed upon them.

As implied by the asymmetry in many domestic broadband installations (down-link has more bandwidth than up-link), the web generally involves collections of users placing requests on a centrally-located server (although now any such 'server' may well be a cluster of physical servers).

One approach to alleviate servers being over-loaded with requests is distributed caching. End users' computers and/or intermediate proxy servers store copies of target web-site data, thus only requiring knowledge that the server data has not changed (so the cache is still consistent) to avoid extra requests being made of the server. Systems requiring very high availability and low latency often now mirror their infrastructure across geographical areas. The advantage of doing so is that the semantics of the application can be programmed into the load-balancing, as opposed to the very simple assistance provided by caching, which is almost completely unaware of target applications.

A more recent development in terms of providing scalable network services is to avoid knowledge of *where* the information is supposed to come from – and instead focus on *what* the information actually is, and how to derive specific destinations from that information. In other words, content-based (or topic-based) message delivery.

A particular advantage of content-based message delivery is that communication between many senders and many receivers is possible. Rather than having to target interested parties, a single message can leave a sender, and be duplicated for each of the recipients closer to the recipients rather than at the server. This means the message load is distributed more evenly across the network infrastructure, and redundant copies of messages can be avoided.

2.3.3 Publish/subscribe systems

The type of content-based message delivery infrastructure we focus on in this thesis is publish/subscribe systems (see [EFGK03] for an overview). Senders of messages are referred to as 'publishers', and receivers as 'subscribers'. Note that messages are often referred to as 'events'. A node may be both a publisher and a subscriber. In large scale systems, the publishers and subscribers are very unlikely to be directly connected, instead they are linked by some path through a network of broker nodes.

Note that the equivalent of broker nodes in the TCP/IP networking described above are routers, switches and bridges. However, for most publish/subscribe research, the broker nodes require more programmability than is possible with current, dedicated network hardware. Thus instead, 'application-level routing' is often employed, where the routing nodes are in fact entire computer systems, and the routing logic is implemented as operating system or application software. Such routing is also common in many peer-to-peer network architectures – e.g. Distributed Hash Table (DHT) [RD01] implementations. Although routing is slower and the components are more expensive, greater programmability and larger memory spaces allow more complex routing logic to enable better management of network failures and dynamic network reconfiguration.

There are various levels of detail at which a publish/subscribe system may examine message contents. The most straightforward publish/subscribe systems will be topic-based (e.g. many commercial systems such as Java's JMS [HBS+02] or Tibco's Rendezvous [TIB99]). In these environments, publishers indicate the topic name under which they are transmitting each message, and similarly subscribers indicate the topics in which they are interested. Existing group communication techniques such as IP multicast can support topic-based pub/sub [Pow96]. For infrastructures that use brokers, each broker in the network will maintain routing state that indicates down which of each of its immediate connections subscribers for each topic reside. Thus if there are no subscribers to a particular named topic, the first broker will drop all messages from any publishers in that topic as there are no targets to which those messages need to be delivered.

Content-based publish/subscribe systems provide much more expressive message routing capabilities at a cost in routing complexity [CRW99]. They can perform routing selectively depending on information contained within every message, and not merely based on equality matching of a topic identifier. Usually in these systems each message (or event) can have a number of named attributes.

The important contrast between topic- and content-based publish/subscribe systems is that the first generally only checks for equality between publications and subscriptions, whereas the latter provides other predicates that can be used for comparison. For example, services monitoring sales occurring in some department store might publish events that include a 'price' attribute. In content-based publish/subscribe, a client may subscribe to events in which the 'price' is larger than some given value. Such subscriptions are stored in the broker state, but now must have an understanding of the underlying attribute types in order to determine how to generalise subscriptions further from the subscribers. As an illustration, suppose a given broker has two clients interested in receiving 'sales' events in our department store; one is interested in sales over £1000 and one is interested in sales over £500. Here the broker must understand that the latter is the more general subscription. Because it covers the other subscription, only the more general subscription need be passed to other broker nodes closer to the publishers.

The Cambridge Event Architecture (CEA) [BBHM95, BMB+00] was an early content-based publish/subscribe system. More recently, the Scalable Internet Event Notification Architecture (SIENA) [Car98, CRW01] proposes a number of network topologies, that of most interest for Internet scale applications being its general peer-to-peer topology, which supports high availability of routes between publishers and subscribers. Leveraging existing JMS technology, the Java Event-Based Distributed Infrastructure (JEDI) [CN01] provides for dynamic formation of event brokers ('event dispatchers' in the terms of this work).

These content-based publish/subscribe systems are not all confined to research – IBM's WebSphere MQ Event Broker [IBM02] includes content-based publish/subscribe features from the Gryphon research project [IBM01].

We do not use all of the features of content-based publish/subscribe in the EDSAC architecture, but do benefit from being able to limit our subscriptions in a number of different attribute fields simultaneously.

2.4 Conclusion

This chapter provides an overview of the key research related to the material presented in this thesis. We begin by describing some of the low-level security tools and data structures we use, and then discuss mandatory and discretionary access control.

Next we introduce Role-Based Access Control (RBAC), and discuss a number of specific considerations such as role definition, hierarchies, groups, parameters, environmental interaction and self-administration. We then examine a number of specific schemes that go beyond RBAC to implement more general role-aware security.

The second section of this chapter provides a brief overview of scalable message delivery systems. In particular we explain the publish/subscribe message delivery model. The EDSAC architecture uses publish/subscribe as its message delivery technology to ensure that the EDSAC architecture is itself scalable.

3 Dynamic privilege management

This chapter describes crucial considerations for the effective implementation of dynamic privilege management. These principles are used as a guide to assess the success of the various case studies we present later in this thesis. We use the term 'dynamic privilege management' to describe access control architectures with features sufficiently powerful to support emerging Internet-scale applications. These principles motivated many design decisions in both the OASIS and EDSAC architectures.

We summarise our ten principles below – the sections of this chapter discuss each of the principles in detail.

1. **Distribution.** Dynamic privilege management and access control technology must operate effectively in a distributed environment.
2. **Dynamic constraints.** Access control systems should be able to support fast revocation of credentials, constraints such as dynamic separation of duties, and applications such as workflow management.
3. **Dynamic credential and capability validity.** Support should be provided for several levels of persistence for credentials and capabilities.
4. **Policy evaluation terminates.** The software that checks for privileges must have well defined termination properties. It is not acceptable for policy evaluations to potentially require unbounded computation or communication delays.
5. **Credential and principal grouping.** For ease of management, credentials and principals must be able to be grouped together. This should include the ability to levy policy over groups of principals.
6. **Environmental interaction.** Whilst the key policy functions of a distributed access control system should be contained within a well defined software body, it is important that dynamic decisions can be made on the basis of conditions outside the access control software itself. Examples include time-based conditions, and external database interaction.
7. **Loose coupling.** There should be a loose coupling between the services being protected by an access control framework and the access control software itself. Doing

so allows the access control system to provide an interface to the policy rules currently in effect, since policy management is delineated from the protected services. This then facilitates evolution of the policy rules without requiring the access control software to be restarted.

8. **Multi-level policy autonomy.** In large-scale organisations, both local and global administration of policy will be required. Generally a number of high-level directives will need to be encoded by all compliant services. However, each service may need to make individual adjustments to the policy they uphold. Access control systems must provide the ability to manage policy rules defined at different levels in a management hierarchy.

9. **Self-administration.** To ease administration in a large-scale system, it is important that the ability to modify access control policy is itself under the control of the access control system.

10. **Audit.** Distributed privilege management systems must provide the ability to record their decision history at adjustable levels of detail to allow later auditing. Such audit activity may be required as legal evidence. It also provides a means to check that the policy itself is compliant with policy administrators' expectations (i.e. for policy testing and debugging).

3.1 Distribution

Increasingly, application environments will consist of distributed, loosely-coupled components operating across networks. The most basic of such applications are web-pages, although web-services are providing a similar style of connection to the underlying computation and data facilities at a site. The next step will be for internal parts of software to themselves run as distributed services. Bennett in [BGL+03] discusses a software as a service (SaaS) system for federating patient record data sources in the United Kingdom's National Health Service (NHS).

Applications are being distributed in this manner to provide flexible deployment structures, resilience to failure and delegated administration. We feel that access control systems should undergo the same transformation.

Distribution appears in many different ways in access control systems. For a start there is a requirement for distributed access control systems to check privileges across the network. In the OASIS architecture, there are two levels of checks. Role-activation conditions are permitted to be comparatively 'expensive' in terms of computation and bandwidth – they are allowed to perform checks that use network interaction. Privilege checks after roles have been activated usually perform 'cheaper', local checks close to the service at which the privilege will be used.

Another primary manner in which distribution should be visible is through policy deployment. It is important that policy checks can be performed based on rules that are stored in various locations throughout the network. The OASIS and EDSAC architectures facilitate this by allowing roles to be activated at a given site and then roles at other sites specifying these former roles as prerequisites.

3.2 Dynamic constraints

As more complex policy rules need to be checked by access control systems, the importance of dynamic constraints will increase. Dynamic constraints involve checking certain policy conditions based on the current state of the access control system itself.

Many of the existing schemes avoid the need to handle such constraints by avoiding conflicts statically. While this is a simpler approach, it significantly reduces policy expressiveness. This can be seen from the sharp distinction between static and dynamic separation of duties constraints. Separation of duties constraints enforced statically ensure that no principal can acquire two conflicting classes of privilege. In an Role-Based Access Control (RBAC) system, this might involve tracing all paths through the graph from principals to privileges formed by the user-role, role-role, and role-privilege relations. Dynamic separation of duties constraints are much less restrictive, instead allowing the principal to acquire potentially conflicting privileges, but ensuring that no conflicts occur between the specific privilege instances acquired. Note that this requires the access control system to be able to make decisions based on its own state of activity.

Another example of a dynamic constraint might be with respect to ‘historical’ conflicts. Here, we want to ensure that once a principal has taken up a position with respect to some target object, we are able to restrict their future privileges related to it. We discussed this type of conflict when presenting the Chinese Wall model (§2.2.1).

We note that many access control implementations provide mechanisms through which limited dynamic constraint checks can be supported. Systems such as OASIS, that permit external database lookups to be performed and then such predicates evaluated over the query results, could store external state for the use of dynamic conflict checking. However we feel this is not a safe, sustainable mode of operation.

One main reason we propose dynamic constraint checking as a primary principle for future access control systems relates to the need for distributed systems support. In a distributed system the state for dynamic conflicts has to be at least partially shared across the network, and there needs to be a communications infrastructure to synchronise participating nodes. Requiring external support for dynamic constraint checking will lead to piecemeal, unsafe implementations that do not integrate easily with other aspects of a distributed access control network.

We acknowledge that dynamic constraint checking is thus more expensive than static checks; our approach in EDSAC has been to provide both mechanisms. Policy authors may choose which is necessary for each policy element in their application.

3.3 Dynamic credential validity

This notion contrasts with older static models for access control, such as the access matrix-based schemes presented in §2.2, in which the relationship between principals and privileges is largely unchanging during operation of the security system.

The need to evolve from static access control models is being driven by systems that require either rapid changes within the access control matrix, or changes in the structure of the access control matrix itself. Changes within the access control matrix will usually relate to the need for fast revocation, or the ability to extend the validity of capabilities already issued. Changes to the matrix structure itself are caused through supporting the dynamic and flexible addition of principals and privileges.

We illustrate some of the main motivating factors for the need for dynamic credential validity below:

Rapid changes in the set of principals. It is important that there is not undue cost when adding principals to a widely-distributed policy system. Indeed to support such a scheme, it is unlikely that all nodes will know the extent of the set of principals at any time. The idea of a global access control matrix only makes sense as an analogy – in fact it will not be possible to capture the state at all nodes in the access control network at some moment in time. In EDSAC we want to be able to support applications in which the set of principals can change more quickly than can be done by the individual administrators of the distributed access control nodes.

Rather than doing reachability analysis from individual principals to given privileges (a useful safety check in access control matrices), it is likely that local role abstractions will instead need to be used for such security analyses.

Probabilistic authentication or trust. Another development in access control infrastructures has been supporting trust assessment in non-binary terms. In other words, some principals may be given privileges without them being explicitly identified to the local decision-making authority. The server will need to balance permissiveness with the risks involved with this permissiveness. In effect some access control decisions become explicitly probabilistic.

Distributed systems may need to permit such 'fuzzy' decisions when entities from other parts of a large-scale federation (such as organisations within the UK National Health Service) authenticate with credentials issued by trusted authorities, but contain foreign attributes not understood by the local service at which these principals are authenticating.

Much access control research has occurred during times when terminal-based login (possibly hidden behind some Graphical User Interface (GUI) session) allowed principals to be identified explicitly. Moves toward distributed provision of services (the most highly cited example being web-services themselves) mean that such rigid ideas of login identity and defined session length are often not applicable.

Whilst not necessarily a web-service itself, consider the example of a web community where a user can create their own login accounts without administrator intervention. If that same user then subsequently logs in frequently, and is 'well behaved' by some metric, it may make sense to automatically increase their privilege level. Of course such a decision would have to weigh up the risk that they are being intentionally deceptive. Such calculations have been explored in the EU SECURE project [DBE^{+}04, Dim03]. For instance, LGI (§2.2.4) has been proposed as a means to broker trust in on-line auction sites.

Applications that require short-lived credentials. Many dynamic access control applications require the ability to support short-lived credentials, and to extend the validity of credentials already issued.

Long-lived credentials are widely used, and well suited to storage in digital certificates (see §2.1.2). Assuming the cryptography in use is secure, these are unforgeable records of the start and end validity times of a credential. However they are thus also highly inflexible. They are not well suited to modifications in their validity times, be that through extension, or more commonly, revocation.

Of course it would be unacceptable if there was no way to double-check the validity of a certificate dynamically – much research has been done into management of revocation within the X.509 certificate infrastructure. The problem is that another digitally-signed credential needs to be issued to confirm the invalidity of the former certificate. Clearly both certificates need to be accessible for proper certification. Thus certificate revocation servers maintaining Certificate Revocation Lists (CRLs) [PHB02] must be accessible for safe operation. To be scalable, revocations must be rare, otherwise all the effort used to digitally sign the truth of the credential in the first place is pointless.

We advocate using an active security network to determine credential validity at the time of use, although acknowledge that a connected access control network will only suit certain applications. Indeed our OASIS and EDSAC implementations also support the use of X.509 certificates for storing credentials whose validity or interoperability is important beyond the scope of our access control architecture. These certificates may be used in the initial stages of authentication with our services.

Workflow management is one application that has a demonstrable reliance on flexible credential validity. Unlike many traditional access control applications, workflow support requires storage of the stage a given principal is occupying currently and suitable modification of the privileges accessible to them. If no deadlines are issued, the duration of credential validity might be highly variable, moreover, once a given stage is complete we may want to stop the principal from reusing privileges valid only to their former task stages. Workflow management usually includes support for team activities. If a number of principals are entitled to perform an action, but it must only be performed once, near-instant revocation of all the other principal's privileges to perform the completed task is desirable. Workflow applications are explored in detail in §7.4.

Note that to support many of the above features, an access control infrastructure must employ some sort of scalable message delivery framework (§2.3). As described in chapter 6, EDSAC uses content-based message delivery in the form of a publish/subscribe system to support dynamic constraints.

3.4 Termination of policy evaluation

In basic access control methodologies, checking whether a principal is authorised to perform some action may be a simple table lookup. More complex systems are likely to involve running some sort of inferencing computation over predefined policy files. This inferencing process may involve communication with other nodes in an access control network, or interaction with external services.

Whilst the range of potential evaluation complexity is wide, it is crucial that the evaluation of policy rules is a process with well-defined termination properties. We do not require that all service calls made within policy evaluations themselves terminate correctly, however, in such cases the caller must consider the service to have timed-out after a predefined period. This then allows the policy evaluation to continue (albeit probably denying the request due to this time-out).

In addition to setting maximum response times for any external service calls, the access control system itself must ensure that its policy rules cannot lead the inferencing process into unbounded recursion or an infinite loop. As a specific example, OASIS policy files permit external services to be called through ‘environmental predicates’. Rule inferencing allows for rule variables to be bound to values from credentials and environmental predicate evaluation. Environmental predicates may also take inputs from the values of variables (for a solution preventing potentially unwanted information flow, see [BEM03]). Otherwise OASIS policy rules are an ordered set of prerequisite checks. To ensure that OASIS policy terminates as expected, it is necessary to check that the prerequisites of rules within policy files bind the values of their variables in an appropriate order. Specifically, the information flow between parameters (via variables), must be topologically sorted, i.e. all variables are bound before they are used themselves.

Note that the environmental predicate calls themselves have no specific time-out behaviour in current OASIS implementations. Adding a time-out can be done simply by making the environmental predicate evaluate to false.

The EDSAC architecture extends this behaviour by explicitly defining timing information bounds in its policy computation. This allows administrators to determine the ideal heartbeat period for the messaging layer that supports fast revocation and dynamic constraints.

3.5 Credential, principal and policy grouping

This principle reiterates much of the logic behind Role-Based Access Control (RBAC); namely for large-scale systems to be practically administrable, there must be support for named abstractions with which to group principals and privileges.

To support the needs of dynamic, distributed access control, we go beyond the basic requirements of NIST RBAC (§2.2.3), however. RBAC mandates that the access control policy defines relationships between roles, and has only edge relationships between roles and either one of principals and privileges. Unlike many RBAC implementations (e.g. [SFK00, NO99, GI96]), we believe that named roles in themselves are not sufficiently expressive to meet the needs of a wide variety of access control applications. OASIS has always supported roles with attributes or parameters.

Beyond parameterisation, we feel that a more general notion of grouping is useful to be applied over the top of RBAC, in particular supporting the grouping of roles as well as groups of principals and privileges. For example, it is likely that only certain groups of roles will require dynamic condition checks; it is important to be able to separate these from the rest, since their policy management is more expensive in terms of bandwidth and latency. As a specific dynamic constraint example, we use role grouping to specify a cardinality constraint: that within a given network only a certain number of roles from the specified group are allowed to be activated simultaneously. In terms of semantics, it may be the case that some roles are explicitly for use at a local site – again it is important to be able to group where the definitions of these roles will reside.

We introduced the notion of *policy contexts* in [BEM04, BEM03]. A particular contrast to the grouping notion of a role is that we permit objects to belong to more than one context simultaneously. Contexts were born out of two independent but complimentary desires. The first was to handle policy evolution, compliance with higher-level policies and distributed administration – in this sense contexts are not used during policy evaluation at all. The other was to provide a means grouping together certain roles for the sake of identifying where dynamic constraint evaluations need to be computed. They are discussed in section 4.4.2.

3.6 Environmental interaction

As previously discussed, the dynamic nature of modern computer communication largely precludes static access control representations as a practical means for policy specification. Sometimes the result of particular policy decisions will only be decidable at the time of request. In an organisation whose access control decisions depend on whether requests are made during business hours, it would be an inelegant and unsafe solution to switch blocks of static access control rules between business-hours and out-of-hours configurations. It is far more practical to design the access control state for the most permissive situation, and apply restrictions at the time of privilege request (this view is also put in [CLS$^+$01]).

This is not merely about the ability to check certain transient conditions immediately prior to role activation or the granting of privileges: careful consideration should be given to how to interface policy decisions with any useful service external to the access control system. In the case studies presented later in this thesis we have explored a number of policies requiring environmental call-out support including database interaction (§5.2), trust calculations (§6.8), location awareness (§7.3), and time calculations (§7.4).

The OASIS system provides a very general mechanism for environmental call-out, which is described in detail in [BMY02a]. In providing any such general call-out mechanism, a number of factors need to be considered, including the following:

Time-outs. As discussed in §3.4, it is important that calls to environmental predicates do not interfere with the ability of policy evaluation to provide a response within a given time period. This will usually mean that time-outs will need to be specified for each external service call, and that the current policy request will fail to be granted if these external service calls respond too slowly.

Information flow. Policy architects must consider that the action of calling environmental predicates provides an information channel from the access control system. If named predicates are likely to be unique in their occurrence within policy files, viewing failed attempts to call them provides information about the current behaviour of principals. Generally we consider this to be a manageable risk – in our deployments we are particularly careful if environmental predicate calls are going to cause network activity.

Further complications are caused by access control systems supporting parameterised policy elements. For example, in OASIS policy, it is possible for any attributes of credentials to be passed to environmental predicates. If these environmental predicates malfunction (through error or malicious intent), it is important for policy designers to be aware of the maximum degree to which information can leak. The use of contexts to assist in the design-time validation of policy is explored in [BEM03].

Cardinality of responses. Another important consideration is potential mismatch in the cardinality of input and output between environmental predicates and an access control system. In OASIS, for example, a number of attributes can be bound to pass information to the predicate, and a number of further variables can be bound from the predicate's response. However, OASIS is not able to handle situations in which multiple environmental predicate response bindings are possible. Say a query is performed on an external database that returns multiple rows. Some mechanism in the environmental predicate must ensure that only one response is returned (e.g. the first possible binding – see §5.2.4).

Back-tracking is not permitted in the evaluation of OASIS rules, to prevent the rule computation latency growing impractical (e.g. if back-tracking causes re-calls of a set of environmental predicates due to the value of their inputs changing). Other access control languages such as Cassandra (§2.2.4) are more expressive in this regard, but also have a more complex rule structure.

Whilst environmental predicates may be extremely useful for increasing the expressiveness of access control policy, it is important to distinguish the design of policy rules from more general programming. As much as possible, policy rules should avoid using environmental predicates to escape the policy inferencing being used – we argue that doing so indicates a weakness in the design of the inferencing process, and will risk causing policy files to become increasingly obscure. See section 5.2.4 for a specific example of how to avoid programming in policy through extensions to the policy language itself.

3.7 Loose couplings

This principle argues that loose couplings are desirable within distributed access control systems, and that loose couplings do not necessarily indicate insecure couplings.

There are two main situations in which we argue that loose couplings are useful:

Policy, infrastructure and application coupling. There should be a loose coupling between the access control policy definitions, the security infrastructure itself, and the applications the access control system is protecting.

A loose coupling between policy and the security infrastructure ensures that policy can be examined and modified independently of the way in which decisions are actually enforced. RBAC epitomises this principle by defining security with respect to names (i.e. roles), rather than keywords specifically tied into application implementation. Java security policies, on the other hand, define privileges over specific classes and methods within a code-base. Whilst this approach is certainly better than hard-coding permission checks, it is less flexible than RBAC.

Permitting policy files to be modified while the security system is in operation allows great administrative flexibility, but does require that version management is addressed. When policy is modified, it must be clear what version of the policy element definitions are currently in use. Further, if a principal is active in some role, the access control infrastructure must correctly handle the situation of that role's definition being updated before the user deactivates it. One simple (but disruptive) approach is to revoke all instances of an active role when a policy update is proposed. A better approach is to require that, where possible, policy evolution is accompanied by a description of the compatibility between different versions of a role definition. This may allow roles activated under the old definition to remain active.

It is also important to have a loose coupling between the access control infrastructure and the application. Primarily this allows restructuring to occur in either without having to keep their operation in strict synchronisation. As a specific example, a number of our test applications could be started and stopped without shutting down the OASIS or EDSAC service being used. If a given access control deployment is properly decoupled and protecting multiple applications, the security infrastructure can continue supporting the other applications and transparently reintegrate the restarted application when it is ready.

Inter-component coupling within the security infrastructure. We also argue that, beyond the couplings between parts of a secured application, the components of the access control system itself should be modular in order to facilitate more freedom in their distribution. Such freedom must be balanced by a focus on the inter-component security of such communication protocols.

For the content-based messaging aspect of EDSAC, we delegate much of the lower-level security to the messaging infrastructure. We have explored aspects of secure publish/subscribe messaging in [BEP$^+$03], and discuss our implementation's loose coupling in Chapter 7.

3.8 Multi-level policy autonomy

For potentially Internet-wide access control systems, it is crucial that the administration is scalable: not just in computing terms but in legal and human terms too.

Whilst in a large organisation global policy will be defined, it is highly likely that local systems will differ in minor respects, and will thus require a degree of local policy-design autonomy. A model of a single policy authority is thus unlikely to be useful. Instead it is likely that *administrative domains* will be defined that agree on certain aspects of policy through *service level agreements*.

Many policy models support a notion of hierarchy that aims to match human resource structures within organisations. We do not believe that it will be possible to fit all policy definitions into a hierarchical structure, though, and thus there must be a means of defining and accessing policy in a distributed manner. There are a number of requirements to support multi-level policy autonomy:

Naming. As for any large-scale distributed system, it is important to manage naming. It may be desirable that role names are stored in a searchable directory. However it is more likely that only services participating in an application will know the names of the roles they wish to activate. Even so, since different applications may be federated over time, it is important that names can be appropriately scoped when necessary. For example, were the EU to mandate federated electronic health record management, various problems might arise combining definitions of 'doctor' roles between the Ireland and the UK National Health Service.

Policy storage. Another consideration in large-scale systems is where the policy definitions are actually stored. This relates to the naming issue discussed above, i.e. how do systems retrieve a definition of a policy element if they only have its globally unique name. In reality, the policy in an access control system is the sum of all the policy rule definitions whether they be global or local. Quite often, however, we refer to the subsets of rules at a particular site as being policy too.

Some distributed access control systems, such as PERMIS (§ 2.2.5), have digitally signed fragments of policy that are directly stored and accessed in Lightweight Directory Access Protocol (LDAP) repositories alongside the credentials of principals.

In such cases there is no need for a notion of policy update; such a process simply involves issuing new policy certificates, and if necessary issuing revocation certificates to annul outdated policy.

In contrast, current implementations of OASIS store policy in local XML files at each service. Thus there is a need to consider how policy update occurs. At the moment, policy evolution occurs outside the scope of the access control environment. It is possible to use OASIS privileges to protect updating of its own policy rules – such an approach is discussed, below, relating to self-administration (also see [Bel04]).

We explore the use of relational databases to store and update such policy files in [BEWM03]. Updates are facilitated by using the active database features in PostgreSQL [SHP89, PD94]. Our primary motivation in using relational databases for policy storage is to take advantage of the existing research into their use in distributed, reliable and consistent storage and update. We did not feel that XML database technology was sufficiently mature to offer the same guarantees of reliability and efficiency for data access and update.

The EDSAC policy representation is more flexible than that of OASIS, combining the best features of both OASIS and the PERMIS approaches, as discussed in detail in chapter 6 and chapter 7. In EDSAC, we achieve this flexibility by preserving the OASIS technique of using XML policy definition files, but by transforming them into Prolog clauses for run-time policy decision making. Our EDSAC environment additionally allows assertion of digitally-signed decision-making clauses into the database if they are from a trusted source, in a manner similar to PERMIS.

Abstraction. As mentioned in §3.6, we found that a number of policy applications could sufficiently fine-tune access control for users' needs without actually modifying the policy rules. However, this was only possible when policy rules contained parameterised elements. This can be viewed as a form of policy abstraction. For example, a policy rule may specify that a certain principal can only use a given role within some hours of the day. Rather than creating rules for each principal, much more manageable policy results when a rule is defined that includes parameters setting the valid day start and end times. Further, when necessary this abstraction can be strengthened by the use of database lookups to allow parameterisation based on each principal.

Sub-networks. To facilitate dynamic constraint checking in distributed access control systems, status messages will need to be sent between access control nodes. A limitation is imposed at this level: attempting to scale up to global coverage would be unrealistic given the very long synchronisation or heartbeat period that would be required for all nodes to remain aware of the current state. Beyond the technical difficulties, such a proposal would not match the organisational structures on the Internet. Current global communication requires the cooperation of many independent network service providers.

We suggest that active access control systems – and thus the messaging systems they use – will need to operate in zones, with gateways federating the separate areas of responsibility. Within each zone, access control messages will have lower latency due to known bounds on their local propagation. Federation of event systems is explored in detail in [Hom02].

3.9 Self-administration

Designing a comprehensive access control infrastructure should include mechanisms for policy evolution. No real deployments will have truly static policy; it is likely that the set of privileges protected will change over time.

In a widely-distributed system within a hierarchical organisation, there will often be a need to delegate limited policy administration to different principals. Not doing so will leave central policy administrators with too much work and responsibility, particularly given they are unlikely to be conversant with the current local requirements.

The need to manage such distributed administration is, itself, a security-related task. In architectures where policy is held in a data-store such as a relational database, it may be possible to use the database management system's security features to delegate update privileges. However, it is more logical to use the access control infrastructure to administer itself as far as possible. Sandhu's ARBAC97 model [SBM99] provides a good example in terms of RBAC. It defines a set of privileges that may be associated with roles for changing access control policy. Extending this self-administration to OASIS is discussed in [BM02b].

Scalable management of widely-distributed systems will be more complex than for centralised policy systems. In ARBAC97, for example, the privileges tend to provide update access to entire RBAC relations. There will be many areas of policy that a local policy administrator in a large-scale policy system will not be permitted to access. In this case, supporting delegated administration requires a way to group policy elements together, and to apply administrative privileges to each of these groups.

One use of policy contexts explored in [BEM03] is to provide a handle for defining scoped administrative privileges. A comprehensive set of administrative privileges suitable for use in most parameterised RBAC systems is described in [Bel04].

3.10 Audit

The final principle we discuss is that of audit trails and log requirements. Any access control system handling large numbers of principals (and particularly distributed administration) should keep detailed records of what privileges have been used by whom, on what and when. Ideally all actions of a system would be logged, but this is likely to be prohibitively expensive, although the specific requirements of each deployment will determine this.

There are three main reasons why audit logs are important:

Verification of correct behaviour. It is unlikely that a large policy specification will be correct and error free when it is deployed. Formal methods could be employed to reduce human error in programming policy, although most of the case studies considered did not employ such techniques – policy files were composed or fine-tuned by humans directly.

An audit trail allows policy errors to be corrected after erroneous access control behaviour has been reported to administrators, or detected by automated tools. Also, as for profiling in other programming languages, reviewing the behaviour of principals in a dynamic access control system means that policy can be fine-tuned to better match modes of observed user behaviour unexpected in the original policy design.

Of course the audit log itself is likely to contain a large amount of highly sensitive data. It is thus important that access to it is tightly controlled; again the notion of self-administration can be applied.

Intrusion detection. Given the increase in the network accessibility of many services, intrusion detection has recently gained significant attention in the access control community. For example, audit logs can be scanned for suspicious patterns of behaviour that may indicate that a principal is attempting to behave maliciously, or has had their authentic credentials compromised.

In large-scale applications, the audit log will grow rapidly. As a consequence, it is unlikely that human users will have the time (or inclination) to browse through the detail of the log records. As in the means for verifying correct behaviour, it is expected that a set of heuristics, such as abnormal numbers of failed privilege requests, would be encoded into automatic log scanning tools.

Legal evidence. The third main argument for an audit log is to help prove compliance to legal requirements. Although the operational efficiency of an application may be greatly boosted by an efficient privilege management system, there is the risk that the paper trail that would have been connected to it before computerisation is lost. Were subsequent legal disputes to arise relating to the use of the application, some record of activity would need to be presented as documentary evidence.

Whilst the motivation for maintaining a detailed audit log may be quite clear, there are a number of necessary technical considerations when designing logging mechanisms for the large-scale distributed access control systems this thesis proposes. Some of these are discussed below:

Distributed logging. In a distributed system logging information is most easily recorded at each participating node. However, it is most usefully recorded centrally, since that way it can be easily seen how the actions of different nodes are related. It may be possible to mandate each node to maintain its own log record for some time period, and 'harvest' these local logs into a central file periodically.

There is also the need to engineer that logs are write-only to avoid falsification of their contents. In systems with centralised logging, it is likely to be easier to guarantee this behaviour.

So far OASIS implementations have generally stored local logs at each service, with their usefulness being focused toward application debugging rather than audit. Because of the introduction of content-based messaging for distributed access control state update, the EDSAC architecture is much better suited to implementing distributed logging. Nodes carrying out audit logging functions can subscribe using a widely-matching pattern. All the messages routed to them can thus be stored. The network heartbeat period provides the time granularity of these events.

If an access control system is very large, or there are particular requirements for reliability or non-repudiation, there will need to be multiple network nodes performing such logging functions. The EDSAC architecture suits such requirements well, simply by fine-tuning the event subscriptions made at each such audit logging node.

Variable logging granularity. As mentioned above, ideally logging should contain as much detail as possible. However, depending on context, the detail level within logs could be variable. For instance, increasing the level of information recorded about the actions of a given principal at some time provides a more detailed body of legal evidence.

In another example, electronic health record management in the NHS, we have proposed that increasing the level of log detail can allow for doctors to perform certain access control system overrides in emergency cases. Such a mechanism is a compromise between upholding all access control restrictions, and not wanting to risk lives doing so.

History-based conflicts. A final consideration that is both technical and policy-oriented is that of avoiding history-based conflicts. In such situations, audit logs may need to play an active part in upholding security. For example, to uphold the constraints discussed in the Clarke Wilson model [CW87] relating to conflicts of interest, it is important to prevent principals from entering into new activities with clients for whom past activities of that principal conflict.

Implementing such historical checks in a distributed system requires that the node at which authorisation is occurring can gain an authoritative declaration of non-conflict from the rest of the network. In such situations the nodes at which audit logs reside can play an active part in the privilege authorisation process.

3.11 Conclusion

This chapter provides a set of considerations we feel are important for next-generation, active, distributed access control systems. We have endeavoured to make sure our requirements are general enough to be applicable to many security architectures. We have also discussed some specific ways in which OASIS and EDSAC relate to the principles we have presented.

The later chapters of this thesis present a number of OASIS and EDSAC case studies. The principles we have outlined above help provide a framework for evaluation for each such study. None of our implementations cover all of the above principles; in that regard our principles also provide important bases for future research into scalable distributed access control systems.

4 OASIS

The Open Architecture for Secure Interworking Services (OASIS – see [BMY02b]), is an access control project that has been running for nearly a decade within the Opera Research Group at the University of Cambridge. Over that time, the computing environment in which it operates, and the technology used to implement it, have changed markedly. Numerous research initiatives have developed it from its original form to the distributed, RBAC, parameterised, policy-based access control architecture it is today.

This chapter provides an overview of the OASIS system, followed by discussion of the major developments during its history. We finish by describing the J2EE CBCL OASIS implementation currently in use.

Each section of this chapter describes both past research, and contributions directly attributable to the research period covered by this thesis. The text emphasises our specific contributions. For simplicity we sometimes attach individuals' names to OASIS developments but in fact almost all major OASIS improvements have been engineered collaboratively with other Opera Group researchers.

4.1 Overview

OASIS is a role-based access control architecture for distributed systems. Its features extend well beyond the basic RBAC models proposed in the NIST and ANSI standards (see [SFK00]), including both long and short lived credentials, appointment (a more general form of credential delegation), fast credential revocation, and a horn-clause logic-based policy language (see [YMB01]) which includes facilities for rich environmental interaction whilst providing bounded evaluation properties.

The OASIS project was begun by Richard Hayton, and documented in his Ph.D. thesis [Hay96]. He described a distributed capability system as discussed in section 4.2.

OASIS roles are activated within a given user session. After initiating a session, the user will be automatically assigned some initial role. Users may acquire subsequent roles and/or privileges by satisfying the preconditions of OASIS policy rules. The structure of these policy rules is introduced in section 4.3.1.

In contrast to most RBAC implementations, OASIS roles and policy rules are managed in a decentralised manner. Each of the distributed objects for which OASIS provides access control is wrapped by a set of rules at a particular OASIS service. These services all operate in an asynchronous manner, and cooperate with each other through the use of an event-based middleware [BMB$^+$00]. This allows effective decentralisation of credential management, since different services can remain responsible for different role prerequisites.

Recent research into RBAC increasingly discusses the need of *context-awareness* for roles. OASIS supports context-aware behaviour in two main ways. Firstly OASIS roles may carry *parameters* (or attributes). Secondly *environmental predicates*, which are also parameterised, may be included in OASIS policy rules. These predicates provide a mechanism through which OASIS rules may depend on local system factors outside the OASIS environment. These two features together allow highly expressive OASIS policy.

Real RBAC implementations often extend the basic RBAC model with notions of delegation and hierarchy. In 'role hierarchies' there is a partial order between all roles in the policy, and more senior roles automatically have access to the privileges of their subordinates. Access control schemes that support delegation must engineer mechanisms to control how often (if at all) the receiver of a delegation may subsequently delegate their privileges.

OASIS does not directly support role hierarchies, since our feeling is that privilege inheritance by superiors in an administrative hierarchy is inappropriate. Indeed, it may be dangerous, for example the managers of engineers in an organisation may not have the technical expertise to use the privileges of those that they manage sensibly. Note that a particular service's policy could always be written to provide the effect of role hierarchies if this was desired.

OASIS supports delegation through the concept of *appointment*, wherein an appointer will present a given appointee (or group of appointees) a particular appointment certificate. Unlike roles, appointment certificates are long-lived digitally-signed certificates, which might be appropriate to express, for example, academic qualification or membership of an organisation. By placing the creation and revocation of these certificates into the scope of access control rules, we allow policy authors to implement the type of delegation which best suits their application.

One of the main strengths of OASIS is its *fast revocation mechanism*. By default, each precondition will be checked for validity only at the time of evaluation of a given rule. However, it is possible to tag any such precondition as a *membership condition*, which means it will be specifically monitored by the OASIS system and must remain valid for the target role to remain active. This is indicated in a rule by tagging the precondition with a superscript '*'.

OASIS services achieve fast revocation by means of so called *credential records*: small structures stored at each OASIS service to indicate their knowledge about the validity of a certain prerequisite. When they believe a prerequisite is invalid, revocation takes place. Due to transitive dependencies, revocation can trigger a cascade of revocations throughout the OASIS network.

4.2 OASIS in 1996

At the time of Richard Hayton's research, the World Wide Web had not developed to offer the sorts of interactivity that e-commerce and web-services have today. As a consequence, most of the inter-operation between sites proposed using protocols, such as the Cambridge Event Architecture (CEA), that were not in widespread Internet use.

The original OASIS implementation focused on the activation and deactivation of names at particular sites. This process required a means to specify the interrelation between these names. The Role Description Language (RDL) was the policy language developed for this purpose.

An OASIS 'role' was defined to be a name that different services had agreed would have particular meaning to that distributed system. Note that RBAC had only just begun to attract widespread academic attention – the first ACM Workshop on Role-Based Access Control occurred in 1995. Even by this time, however, OASIS roles were parameterised entities, and the RDL was a highly expressive policy language.

4.2.1 The Role Description Language (RDL)

RDL was the first policy language developed for OASIS. A 'role' is defined as a name with a list of arguments:

$$\text{def } \mathbf{Rolename}(arg_1, \ldots)\ arg_1 : \text{type } [arg_2 : \text{type} \ldots]$$

Access control policy is effected by a set of rules at a given site. Each of these rules is in one of three forms:

Role entry.

$$\mathbf{rname}([arg_1, arg_2, \ldots]) \leftarrow \mathbf{prereq}_1\ [\wedge \mathbf{prereq}_2 \ldots]\ [:\mathbf{Constraint}]$$

This rule indicates that to activate the role **rname**, the requester must already be active in all of the $\mathbf{prereq}_n$ roles, and their parameter values must satisfy the given **Constraint**. Note that the arguments to role **rname** must be included in the order they were specified in the role definition, although possibly with different variable binding names (although this is not made clear). The constraint language defined provides numerous value comparison, boolean, and set operators [Hay96, p21].

Election. These rules extend the role entry form of rule, by including an extra term:

$$\mathbf{rname}([arg_1, arg_2, \ldots]) \leftarrow \mathbf{prereq}_1\ [\wedge \mathbf{prereq}_2 \ldots] \lhd \mathbf{elector}\ [:\mathbf{Constraint}]$$

In this case a principal independent from the requester, and currently active in the role **elector**, must indicate their support for the role request to be successful.

Revocation. Finally, the revocation rule has the opposite effect of an election rule – principals active in the role **revoker** may request the deactivation of the role **rname**. The form of revocation rules is as follows:

$$\mathbf{rname}([arg_1, arg_2, \ldots]) \leftarrow \mathbf{prereq}_1\ [\wedge \mathbf{prereq}_2 \ldots] \rhd \mathbf{revoker}\ [:\ \mathbf{Constraint}]$$

The role-based revocation proposed was not actually implemented or tested in Hayton's work, as he makes clear in his dissertation.

Another key OASIS notion was introduced in RDL, namely that of *membership conditions*. If a prerequisite role or condition in the above rule expressions appears with a superscript asterisk (e.g. $\mathbf{prereq_1^*}$), this condition must remain true if the target role is to remain active.

Given OASIS operates in a distributed environment, membership conditions require that the access control infrastructure has a mechanism to notify remote services when the status of credentials change. Hayton acknowledges this, and includes a chapter describing an event-based communication paradigm that could support fast revocation, but he does not explicitly specify how to integrate these technologies. One of the primary contributions of the EDSAC architecture is its provision of scalable fast revocation, and a number of new policy features supported by more recent event-based communication research (see chapter 6).

4.3 OASIS up until 2002

Hayton's thesis presented a policy language (RDL), the way in which an implementation might record credentials, and the event paradigm it might use. However all these aspects were introduced somewhat independently. Work soon began on formalising the process of OASIS policy inference.

4.3.1 Formalising the inference process for OASIS rules

Following from Yao's presentation of [YMB01], a comprehensive description of the semantics of OASIS rules was presented in [BMY02a]. We only present the change in notation from RDL here, and highlight salient contrasting points.

Given the increasing focus on RBAC, OASIS compared favourably to many alternative schemes because of its parameterisation, and expressive policy language. Where RDL had allowed various forms of initial roles, OASIS was developed to activate OASIS roles explicitly in the context of a user session. Also, whereas RDL had mostly focused on whether roles were active or not, the OASIS policy language was extended to explicitly manage privilege requests (in line with other RBAC research going on at that time).

Role-based revocation and election forms of RDL were also removed from the core language syntax, since their function in policy rules had been replaced by *appointment* *certificates*. As mentioned above, the action of appointment involves a principal using privileges that create and revoke credentials to be used as role entry prerequisites by other principles. Whilst this notion appears similar to RDL election, the contrast is that appointment certificates explicitly exist beyond the duration of OASIS sessions. To allow delegation in RDL, it appears both a delegater and their target would need to be active in roles for some overlapping period of time. Appointment removes that requirement.

Two types of OASIS policy rule were developed. *Role activation rules* check all conditions are satisfied before roles are activated within a session. *Authorisation rules* then perform further checks before privileges are exercised on a protected system. Note that these two rule types relate to the fundamental RBAC principal-role and role-privilege mappings respectively. Since OASIS roles are parameterised, so are these rules. Unlike RDL, the new OASIS policy syntax merges the **Constraint** terms and the prerequisite conditions.

This merging of terms clarifies the order of inference processing (left to right), and as well as appointment certificates, also allows the environmental predicates to be included to evaluate conditions external to the OASIS access control framework (solving problems such as Hayton's desire to support external ACL functionality).

The basic structure of a role activation rule is as follows:

$$r_1, r_2, ..., r_{n_r}, ac_1, ..., ac_{n_{ac}}, e_1, ..., e_{n_e} \vdash r$$

The r_i, ac_j and e_k terms represent the n_r prerequisite roles, n_{ac} appointment certificates and n_e environmental constraint predicates in this rule respectively – note that it is acceptable for any of n_r, n_{ac} or n_e to be zero, provided at least one is non-zero. Predicate expressions on the left hand side of the rule are called preconditions, and must be valid for a given user to activate r, the target role. Roles and appointment certificates are valid if they have not been revoked. Environmental predicates are valid if they evaluate to be true.

Authorisation rules are of the following form:

$$r, e_1, ..., e_{n_e} \vdash p$$

There is one—and only one—prerequisite role r. The environmental constraints e_k behave as for role activation rules, and, finally, p is the target privilege of this rule. A set of the above role activation and authorisation rules defines the policy for a given OASIS service.

4.3.2 XML policy and an Apache OASIS module

Both RDL and the newer OASIS use syntax inappropriate for policy representations in operational systems. Yao's implementation was the first to use an XML representation for OASIS policy files. The syntax of the CBCL OASIS policy files is slightly different from Yao's OASIS, but the basic structure remains the same. We explore the XML policy representation in later chapters when explaining how our extensions to it take advantage of newer EDSAC capabilities.

Yao's implementation of OASIS was built as an extension module to the open source Apache web server [ASFa]. This module managed OASIS sessions at a given service, and responded to role activation and privilege requests appropriately. Unfortunately, a lack of documentation, and some intellectual property issues, left Yao's OASIS implementation unused.

4.3.3 OASIS infrastructure protection

On a more theoretical level, during the time Yao was reimplementing OASIS, we explored the need to protect the underlying OASIS network infrastructure from attack: see [BE03] (originally presented in 2002). Our work followed from Hayton's discussion of how service nodes in an OASIS network should behave when the connectivity heart-beat messages are lost.

We examine three areas in which to defend against attacks against the OASIS network infrastructure itself, as described in the sections below.

Heartbeat failure

Each local OASIS service stores its belief about the state of remote services in a *credential record*. When a heartbeat message is missed, the relevant external-credential records are annotated with the special tag *unknown*, until the heartbeat resumes. Unknown states may trigger cascading revocation. To combat malicious users attempting to cause denial of service attacks by perturbing the heartbeat messages, in [BE03] we propose extra tagging of membership preconditions in a rule to avoid immediate cascading revocation.

Dependency estimates

The second infrastructure protection scheme we propose relates to policy analysis. OASIS policy may be visualised as an acyclic, directed graph. The nodes in this graph are the roles and appointment certificates defined in the policy file. The edges of the graph connect prerequisite credentials to their target roles. Actual policy evaluation also includes environmental predicates, that are effectively additional guards on these edges. This graphical visualisation of policy is explored further in chapter 5.

We propose calculation of *dependency estimates* as a tool for static policy analysis. We do not consider environmental predicates in these estimates, and membership conditions are not weighed any differently from other rule preconditions.

Consider some appointment certificate a. Let us define the set $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ to be all rules that include a in their left hand sides. A crude measure of dependency might simply be the cardinality of $\mathcal{R}$. Clearly this ignores transitivity.

Instead we suggest taking the set $\mathcal{T} = \{t_1, t_2, ..., t_n\}$ formed from all of the corresponding targets of each of the roles in the set $\mathcal{R}$. The dependency estimate of a is then $de(a) = \sum_{i=1}^{n} de(t_i)$. In other words, we sum the dependency estimates of the target roles based on it. Defined recursively, $de(t_i) = \sum_{j=1}^{m} de(t_j)$, for each role t_j dependent on t_i. The base case of a role t_k on which no other role depends is $t_k = w$ for some weight factor w. We suggest that terms in $\mathcal{R}$ and $\mathcal{T}$ that might cause cycles be filtered from the summation in the preceding equation.

These dependency estimates will provide policy administrators with an indication of the critical preconditions within a given OASIS domain.

Threshold-based rule evaluation

A further suggestion was to extend the expression of OASIS rules to support multiple-party, weighted voting, or hand-over policies. Similar schemes are proposed in [KS99, San88]. Under threshold-based evaluation, OASIS rules would be expressed as follows:

$$r_1 \cdot x_1, r_2 \cdot x_2, ..., r_{n_r} \cdot x_{n_r}, ac_1 \cdot y_1, ..., ac_{n_{ac}} \cdot y_{n_{ac}}, e_1 \cdot z_1, ..., e_{n_e} \cdot z_{n_e} \vdash_w r$$

where $\sum_{i=1}^{n_r} x_i + \sum_{i=1}^{n_a c} y_i + \sum_{i=1}^{n_e} z_i \geq w$ must hold for this rule to evaluate true. If these weight factors are omitted from the rule, implicitly x_i, y_i and z_i values are taken be equal to one, and $w = n_r + n_{ac} + n_e$.

The rule $a{\cdot}3, b{\cdot}1, c{\cdot}1, d{\cdot}1 \vdash_5 r$, can represent an example of multiple-party appointment. In this case, let us assume that a is an initial role, and b, c and d all represent appointments regarding the same qualification for this particular user. By setting the threshold weight at 5, we have effectively specified a policy whereby a two-thirds majority of the appointing parties is acceptable. Note that there is an underlying separation of duties constraint here – any of the appointing parties must not be implicitly permitted to act as any of the other appointing parties.

This extension of rule evaluations can also assist with a number of legitimate problems that could possibly cause appointment revocation, for example appointment hand-over or emergency role activation. Consider the rule $a \cdot 3, b \cdot 1, b_{override} \cdot 1, a_{emergency} \cdot 4 \vdash_5 r$. In the case of appointment hand-over, the party that originally provided appointment b may be phased out of the OASIS system. To make sure we do not get unnecessary revocation, a super-user may enable the $b_{override}$ appointment certificate temporarily until the hand-over target party can provide a new b appointment. As an overall measure, the $a_{emergency}$ role requires only one other precondition to activate role r.

Bounded session durations

Our final suggestion for increasing the security of OASIS infrastructure involves bounding the duration of user sessions. All user sessions are tagged with an explicit lifetime duration attribute that is propagated to the membership records dependent on this session. The CBCL OASIS implementation discussed in section 4.4 uses such a session-duration limit. The imposed duration will depend on the OASIS service and the initial roles associated with the session in question. The lifetime granted by a server might change as statistical data is accumulated about the system's roles and their usage.

4.4 OASIS after 2002

In 2002 the first comprehensive implementation of OASIS was completed. The design of this implementation and the Electronic Health Record application it supports was done in collaboration with Clinical and Biomedical Computing Limited (CBCL). CBCL are a Cambridge company with very strong connections to the Clinical and Biomedical Computing Unit (CBCU), which itself fits within the Centre for Applied Learning and Educational Technology (CARET). Since the coding of this implementation was largely performed by CBCL, we refer to this code body as CBCL OASIS.

4.4.1 Java 2 Enterprise Edition (J2EE) implementation

CBCL OASIS is a Java 2 Enterprise Edition (J2EE) [Sha01] application. The J2EE infrastructure provides a large number of services useful for building web-based server applications – Sun's J2EE technology is often contrasted with Microsoft's .NET architecture [Mic02]. For state persistence, all J2EE application servers provide a relational back-end database. CBCL OASIS uses the Orion (1.6.0 beta) application server [IF01], and a PostgreSQL [SK91] back-end database. CBCL OASIS sessions involve the creation of Enterprise JavaBeans instances – if the server loses its memory, when restarted, it will restore the old OASIS state from the database.

The J2EE infrastructure provides an HTTPS server. CBCL OASIS uses HTTPS both to communicate via HTML with user browsers, and to exchange SOAP messages with other CBCL OASIS sites securely. J2EE supports Java Server Pages (JSP) [Sun99] that allow customisable server document generation. JSP pages are used to present web-pages customised on the basis of what OASIS privileges are accessible to that particular viewer.

Overall, the CBCL OASIS implementation is ideal for creating web-based OASIS protected applications. Unfortunately the J2EE architecture is very complex. The J2EE API is comprehensive, each application server provides different features in different ways, and there is virtually no documentation of the actual CBCL OASIS source-code. The CBCL OASIS code was built with speed as a priority – there are a number of rough edges and hard-wirings still present that greatly complicate installation. Nonetheless, it was used to successfully built both of the case studies discussed in chapter 5. CBCL OASIS uses an XML policy representation very similar to Yao's OASIS implementation. Examples of this language are provided in following chapters. Note that CBCL OASIS validates policy using a Document Type Definition (DTD) [W3C04] rather than the more heavyweight XML Schema [W3C01b, W3C01c, W3C01d].

Differences from earlier OASIS designs

CBCL OASIS differs from earlier OASIS designs in a number of ways. Users authenticate and gain an initial role through X.509 certificates, using client-side SSL. User sessions are directly linked to these X.509 certificates. However, unlike previous implementations, user session management is implicit. When a user connects to a page, an OASIS session will be created for them if one does not already exist. Further, this session will only be terminated if it is left inactive for a given period of time.

Role certificates are not digitally-signed credentials in CBCL OASIS. Because OASIS services need to establish trust in each other in order to form inter-service SSL links, the extra cryptography was deemed redundant. Instead unique OASIS *tokens* represent user sessions. Also CBCL OASIS does not provide a system-level heartbeat protocol; its focus is generally higher-level than Hayton's original system-level design.

Using J2EE itself represents an overall difference in implementation philosophy. J2EE, as the acronym expansion suggests, is most often used for the development of industrial applications. The server load-balancing features included in application servers make them very useful on high-use web-server clusters, for example. Generally most academic projects implement prototypes designed to facilitate future research. Indeed our EDSAC implementation was explicitly designed to be much more light-weight than CBCL OASIS for this very reason – researchers do not want to invest time in learning how to develop a particular type of software; their interests lie in the function of the software rather than its implementation.

4.4.2 Policy contexts

During the development of CBCL OASIS, it became increasingly clear that OASIS policies needed a mechanism for classification. The XML schema for OASIS policy was focused on representing rules locally at each particular service. Clearly in large-scale policy deployments better naming and scoping would be required.

In collaboration with Belokosztolszki, we developed *policy contexts*. We first presented our ideas in [BEM03], and refined the formal model for them in [BEM04]. Our later work increases the scalability of contexts by specifying a hierarchical naming structure for them.

Contexts provide a means to label policy elements. By policy elements, we mean OASIS roles, appointment certificates, rules and privileges. In addition, parameters of roles, appointment certificates, and environmental predicates can also be labelled. To make the classification as expressive as possible, each policy element may have multiple labels. These labels are *context elements*. A set of such elements forms a *context*. The mathematical syntax introduced involves the label tags as superscripts to policy elements – each Greek lower-case letter indicates a context (i.e. a set of context elements). For example:

$$\text{rulename}^{\alpha}: \qquad r_1^{\beta_1}, .., r_{n_r}^{\beta_{n_r}}, ac_1^{\gamma_1}, .., ac_{n_{ac}}^{\gamma_{n_{ac}}}, e_1^{\theta_1}, .., e_{n_e}^{\theta_{n_e}} \vdash r^{\mu}$$

Where α is a rule context, β_i and μ are role contexts, γ_j are appointment certificate context tags, and each θ_k is a context tag for an environmental predicate. It will be more useful in actual policy specifications to tag a policy element with its context element labels rather than the context as a whole. For example, $r^{\{\texttt{intranet\_only,personal}\}}$, rather than r^{α}.

Contexts may be used for both policy administration (as discussed in [Bel04]), or for levying dynamic constraints over the policy elements within any given context (as discussed in chapter 6). Example uses include:

- Audit detail can be context dependent.
- Contexts can classify organisational and functional roles.
- Access control to the policy file itself can be policy context based – to support distributed administration, for example.
- Cardinality and dynamic separation of duties constraints can operate on the sets of policy elements classified into certain contexts.
- Environmental interaction can use context-tagging on predicate parameters to indicate where potential information flows exist in and out of the access control system.

Information flow between context elements

Information flow analysis has been frequently applied to computer security. For example, Mandatory Access Control (MAC) schemes usually enforce some form of information flow constraint over their security lattices. Denning provides an overview of many such systems in [Den76]. Recent examples of RBAC policy information flow analysis includes [Cho04], in which RBAC is used to support security in object oriented programming. Jaeger [JEZ02] presents a policy segmentation scheme that achieves some of the administrative objectives of contexts. Context applied to authorisation policy is explored in [McD03].

Bidan and Issarny model information flow as an extension to access control [BI98]; they analyse the information that can flow from objects back to the subjects permitted access.

Policy contexts provide a mechanism through which we can analyse the information flow properties of OASIS and EDSAC policies. Details of our information flow formalism are presented in [BEM04].

4.5 Conclusion

This chapter presents the OASIS access control architecture, and traces its development from its initial description in the work of Hayton to the present day. The past decade has seen numerous implementations developed to various degrees of completeness. Over time the policy language has evolved, and the server and application architectures have changed significantly.

We have also indicated a number of areas in which our collaboration with other members of the Opera Research Group and CBCL has led to development of OASIS, including in the design of the CBCL OASIS implementation. CBCL OASIS is the basis of the case studies presented in the following chapter.

5 OASIS case studies

This chapter examines the design and implementation of two OASIS deployments. Both deployments led to developments of the OASIS architecture. We highlight our contributions within the sections below.

The first case study (§5.1) describes the design and implementation of an electronic health record (EHR) interface for the United Kingdom National Health Service (NHS). OASIS is used to enforce access control policy. In the second case study (§5.2), OASIS is used to manage roles and privileges in an inter-institutional electronic courseware system.

5.1 Access control for Electronic Health Records

The United Kingdom NHS strategy aims to have a full electronic health records service available nationally by 2008. Secure, scalable, access control infrastructure will be critical to the success of such an initiative. This section presents our prototype for such a service, built using the OASIS architecture.

Providing electronic health records (EHRs) with high availability, yet maintaining their protection from unauthorised access is a complex yet crucial task. One goal published within 1998 United Kingdom National Health Service (NHS) Information Technology strategy documents [NHS98] was to achieve EHR support nationally by 2005 (or earlier). This highly ambitious target was revised in 2002 to suggest national EHR support by 2008. Even so, we feel it is unlikely that the full scope intended of NHS EHRs will be achieved by then.

Access control in EHR systems poses a number of challenges not faced in other security environments. The information being protected is highly personal – security breaches may lead to irrevocable consequences for the individuals involved. Even so, there is a need to monitor the state of the NHS. Thus it is likely that certain types of anonymised statistical summaries will be required.

Another difficulty is heterogeneity – both within management and at the system level. Different Health Care Organisations (HCOs) will all have slightly varying procedures, and thus varying policies dictating access control. These local differences will need to be balanced with the overall directives of the NHS.

Perhaps the most challenging aspect of the NHS EHR strategy is the amount of power given to the patient in terms of access control. ‘Patient consent and confidentiality’ are both considered key issues relating to the NHS EHRs. This amounts to patients being able to specify fine-grained access control restrictions over their individual EHRs if they so desire – potentially leading to a large, distributed, complex policy.

To further complicate matters, there are various situations in which health professionals will need to apply emergency overrides. In simple terms, patients may unknowingly specify access control rules that are not in their own best interest, and which may indeed turn out to be life-threatening. Thus an expressive policy language is required to manage these policy conflicts.

5.1.1 Related research

Before we describe our CBCL OASIS EHR prototype, we examine some of the other work in the field of EHR access control. There have been many architectures proposed to handle EHR systems. One early project that does not focus on security, but has a similar network architecture to our EHR prototype, is Synapses [GBG^{+}96].

The Patient Centered Access to Secure Systems Online (PCASSO – [BBB97, MBBC02]) project is a trial EHR system run in the USA. Although it claims to be role-based, its role classifications are very basic. Overall the project focused on security risks in the computing infrastructure (e.g. combating Internet café web-browsers logging keystrokes and data) rather than the complex policy issues required to support patient control over patient records.

One potential approach to managing policy conflicts arising from medical emergencies is the Tees Confidentiality Model (TCM) [LLN03b, LLN03a, LLCT00]. It provides a specific ordering designed to resolve conflicts consistently with the proposed UK Patient Confidentiality Requirements.

In the TCM, confidentiality permissions are processed in a defined order, connected with specific types of policy from least to most powerful (in terms of override) labelled: IRC, ISC, IGC, RSC, and RGC (the letters I, R, S, G and C representing Identity, Role, Specific, General and Confidentiality respectively). The idea is that these stages cover all the useful configurations for positive and negative permissions levied over individuals, groups of individuals (e.g. teams), roles, groups of roles, particular data records, and groups of data records.

Since the TCM is very specifically engineered to the medical context, it is likely that its features could easily be included into other access control technologies. Although it includes negative permissions, which complicate authorisation inference, it does so in a strictly limited manner among the above override categories.

The PrivilEge and Role Management Infrastructure Standards validation (PERMIS) system (see §2.2.5) has also been applied within the health care domain, as described in [CM03]. In that study, PERMIS facilitated a secure, distributed system for the issuing and collection of prescription medication.

5.1.2 An architecture for NHS electronic health records

This section presents the design of our NHS EHR infrastructure, and discusses how our prototype implementation demonstrates its applicability. The subsections below describe major design considerations.

Heterogeneous services

In an ideal world, requirements analysis could be performed over the entire EHR domain, an implementation written, and health care organisations (HCOs) be required to change over to it. Of course this is completely unrealistic. For a start, the requirements of health care data protection will always continue to evolve. Take, for example, how HIV status has complicated EHR policy – in such cases, it may well be that the patient would want positive HIV status to remain as confidential as possible, yet many hospital procedures will need to take extra precautions in the management of HIV-positive patients.

Given constantly developing requirements, and financial and time constraints, different parts of the larger HCO network will update their software without global synchronisation. Our EHR infrastructure is designed assuming we will have to integrate heterogeneous data sources.

One particular advantage of the NHS context is that fields such as NHS identifiers are almost invariably present, and are used in a semantically consistent manner in EHRs. Of course high-level medical training will often be required to grasp the combined effect of these EHRs. These observations form the basis of our 'multi-window approach', a term introduced by CBCU director Jem Rashbass during the EHR design process. Our approach involves EHR fragments being retrieved from different sites, and the joining of important information being done by qualified practitioners viewing these fragments, rather than by an automated process. Thus we support the management of heterogeneous sources by retaining an active human role in the merging of the data they may individually present.

Using simple attributes such as the NHS identifier (ID) of the patients and those of the practitioners treating them, combined with the time-stamp of the event in question, an index can be formed over EHRs without the need to become involved in the semantics of the events in question. This then avoids the need to define a centralised data dictionary of terms and messages.

Another advantage of this thin indexing approach is that it will allow a global index to be formed that maintains very little data per record. A Uniform Resource Identifier (URI) references the bulk of the medical data at the hosting HCO sites. Given the index will span a large number of active records, it is desirable that this index should have minimised resource requirements.

One disadvantage of our approach is that because there is a two-step process in retrieving record fragments, the access control policy cannot be specified at the level of data used in the multi-window retrieval, and instead needs to be levied at the HCO EHR fragment level. Whilst administration of the policy becomes more complex, an advantage is that it is easier to guarantee that access to particular EHR fragments will be denied. More difficult is ensuring the availability of aggregated results. Note that it is also possible to use the index service as a point of anonymity support since it can broker a transaction between a client and an HCO using pseudonyms. This is discussed in section 5.1.3.

The hierarchical nature of our design also resonates well with the way policy is likely to be specified within the NHS. For example, there will be high-level policy directives connected with the Integrated Care Records Service (ICRS) that operate over the entire NHS, yet each individual HCO will need to tune policy to suit their specific organisation.

Conversion of data from legacy schemas is an extremely difficult problem well studied in database research. Whilst two particular HCOs may store the same information in their databases, there are two classes of difficulty in combining information from them. Syntactic heterogeneity involves the same information represented in two isomorphic ways. For example a time-stamp might be represented as the time of day on a given day of a month of a year (to second accuracy), alternatively it could be represented as a number of seconds after some particular date. Semantic heterogeneity is more complex to correlate – for example contrast storing an address using separate house number, street name and suburb fields, with instead using Global Positioning System coordinates.

We impose an overall network infrastructure, but require each service to only interoperate on a limited set of terms. These terms can facilitate opening channels to the HCOs that show web-based EHR fragments to a medically-qualified user, and thus avoid the need for complete standardisation of the data dictionaries in use at each HCO.

An NHS EHR infrastructure

The main components of the CBCL OASIS prototype are presented in this section, and are illustrated in Figure 5.1. It is important to note that only the Index Service is a single logical object, although we expect it would be replicated for the sake of availability and reliability. Each of the types of node in the figure is described below:

HCO Server. It is assumed that each health care organisation needs to have an OASIS-aware service that allows it to interact with the rest of the EHR network. If a legacy application is being used at a given site, the OASIS-aware service must define sufficient local policy to allow this application to participate in the wider EHR network.

Ideally HCO servers would be natively OASIS-aware although, as mentioned above, it is unrealistic to expect all sites to be in a position to deploy a new software infrastructure. Instead, they can apply an OASIS-aware interface wrapper.

Given HCO servers contain the detailed EHR information, the most semantically rich access control policies will be enforced by the HCO servers.

Index Service. The function of the index service is implied by its name. It stores a rudimentary header for each EHR fragment, and which HCO to contact for the complete record. It also translates the source of requests for EHR fragments into pseudonyms before they reach the HCO sites in question.

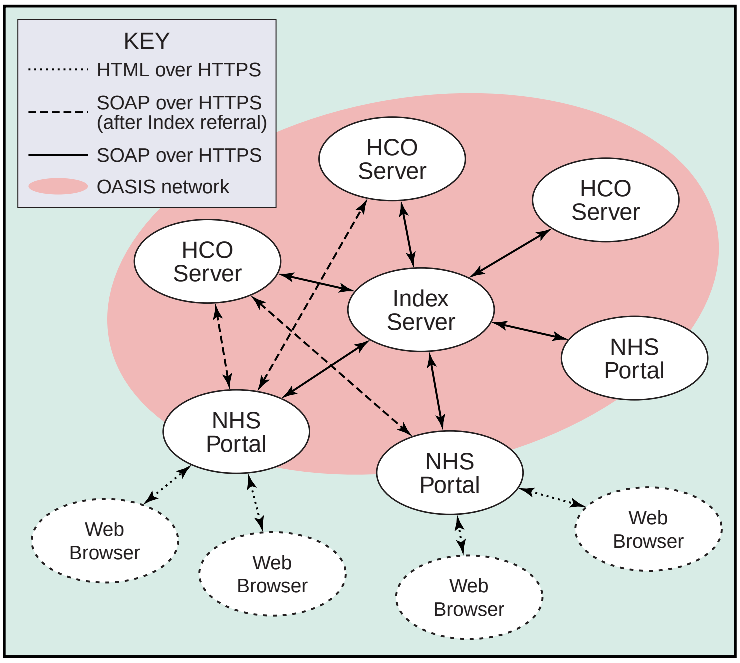


Figure 5.1: Our EHR network components

Note that it is possible for the index service to contain access control policy. For example, certain situations may require that even the EHR headers are protected from certain requesters. As an instance, some HCOs may through their name alone imply sensitive information, such as psychiatric institutions, or HCOs related to blood testing or HIV/AIDS – in such cases Index Server policy might filter responses based on the NHS ID of the medical practitioner.

NHS Portal. The NHS Portal allows patients and/or medical staff to connect to the EHR network. The current prototype provides an HTTPS (i.e. secure) web-serving ability so that users' terminals need only run a web browser to allow them to use the EHR network (which is an OASIS service network). Numerous users can be managed by any given NHS portal, and NHS portals themselves can be replicated to provide greater efficiency (e.g. geographical localisation), and increased reliability.

Again, the NHS Portal may also contain OASIS policy. This avoids expensive rejection (in terms of time and communication cost) by the HCOs in cases where a fairly obvious policy breach is being submitted.

Communication between the above components occurs in both internal and external ways from the perspective of CBCL OASIS. Internal communications occur between the software at HCOs, the Index Server, and the NHS Portal. Internal communication transmits Simple Object Access Protocol (SOAP) messages [W3C03a, W3C03b, W3C03c] over HTTPS connections. The external communication between users' web browsers and the NHS Portal currently uses conventional HTML over HTTPS.

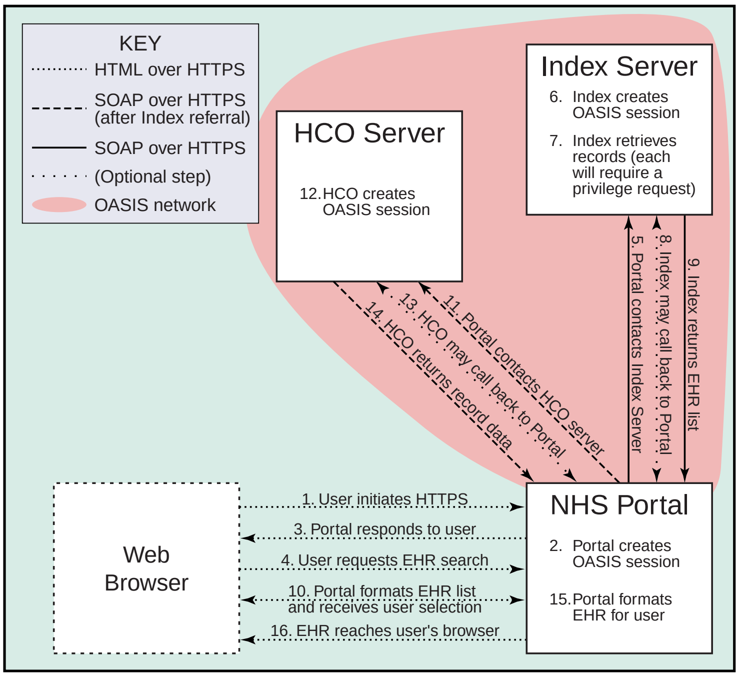

KEY
HTML over HTTPS
SOAP over HTTPS (after Index referral)
SOAP over HTTPS
(Optional step)
OASIS network
Index Server
6. Index creates OASIS session
7. Index retrieves records (each will require a privilege request)
HCO Server
12.HCO creates OASIS session
11. Portal contacts HCO server
13. HCO may call back to Portal
14. HCO returns record data
5. Portal contacts Index Server
8. Index may call back to Portal
9. Index returns EHR list
NHS Portal
2. Portal creates OASIS session
15.Portal formats EHR for user
Web Browser
1. User initiates HTTPS
3. Portal responds to user
4. User requests EHR search
10. Portal formats EHR list and receives user selection
16. EHR reaches user's browser

Figure 5.2: Retrieving an EHR using the CBCL OASIS NHS prototype

An example interaction between a user and a health care organisation

To put the interactions between the various CBCL OASIS components (all of which are OASIS-aware services) into context, we provide a trace of the overall steps taken between a user making a request for EHR data, and a formatted response arriving back to that user. The steps in this communication are shown in Figure 5.2, and are described below.

Note that before users can communicate with the NHS Portal, they must have been issued with an X.509 authentication certificate. Because this step is likely to occur once, long before users make EHR requests, and because the distribution will be out-of-band with respect to OASIS, we have not included it in the list of steps below.

The user's X.509 certificate contains information used as the initial appointment within the user's OASIS session. Initial role acquisition is thus split into two phases; opening an initial connection, and providing initial credentials. For a communication channel to open successfully, there must be a common certificate on the certificate chain used by the client and the NHS Portal. Extension fields within this X.509 certificate allow the NHS Portal to issue a starting role within this particular user session.

The CBCL OASIS prototype uses a web-based system for issuing certificates. Whilst this was not the focus of the prototype, and thus there are security needs not addressed in the development of the certificate generator, modern web-browsers do much of the certificate management automatically on behalf of users – they will correctly store a client-side certificate presented to them with no explicit user direction. The process does require some user interaction, but only to the extent users must affirm prompts from a 'wizard' performing the certificate import process on their behalf.

There are two broad categories of access we would expect to the NHS Portal; people retrieving their own health records, and medical or administrative staff retrieving EHR fragments to which they have some official connection. In the example below we trace through the interaction we would expect for a patient. We use 'patient' or 'user' to indicate the human user involved in this interaction, and 'client' to indicate their computer. The 'server' being discussed should be clear from context.

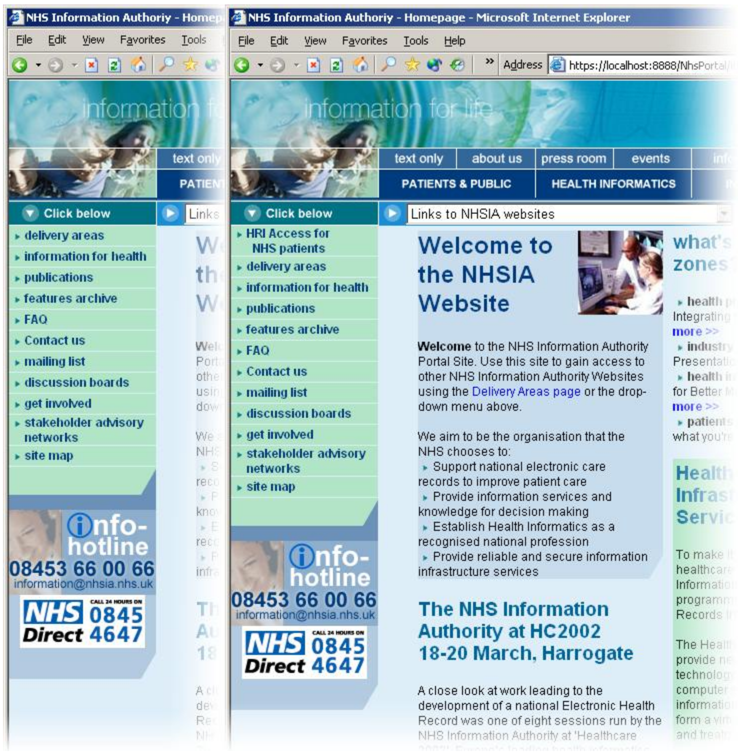
1. Our patient decides they wish to examine one of their EHRs. The first step they take is to open their web browser, and direct it to the NHS Portal site. This site uses HTTPS security. Note that unlike most e-commerce sites, the underlying Secure Sockets Layer (SSL) connection requires both client- and server-side authentication. The server-side authentication is common; the NHS Portal in this case expects to need to present its server certificate. Client-side trust is established automatically by the client browser if the server's certificate was issued by one of the root or intermediate certification authorities pre-loaded into the user's browser.

   Client-side authentication will usually be done automatically if the client only has one suitable certificate in their browser's key-store (i.e. only one of their X.509 keys permits authentication usage). Otherwise, the user is asked by their browser to select from a list of the certificates they have available.

   In the cases where the user is not simply a patient, it is likely that they would see client-certificates for both their role as a doctor or administrator, and also for their role in the NHS as a patient. Note that these multiple certificates could be replaced by an identity certificate, and the policy file use other techniques (such as database lookup) to present them, after authentication, with a choice of role. It has been observed that the CBCL OASIS implementation currently blurs identification and attribution functions in its X.509 certificates. There is definitely identity/attribute overloading at the moment, although it could easily be addressed if required.

2. After successful mutual client/server SSL authentication, the NHS Portal site creates an OASIS session. In CBCL OASIS this involves the creation of an Enterprise JavaBean (EJB) that is added into the back-end store so that server interruptions (e.g. power failures) will not cause the session state to be lost. In particular, the Structured Query Language (SQL) table used to make EJB state persistent contains a long integer identifier for this session. We refer to this ID as an OASIS 'token'. The token is essentially random, in that there is no explicit correlation between the user and the token.

   In the current CBCL OASIS implementation, because there is no way to know whether the user will make further requests of the NHS Portal within this session, a session timer is also initialised. When this timer expires, the session is automatically removed. Further interactions from the user may warrant resetting the timer to extend the maximum session duration.



Welcome to the NHSIA Website

Welcome to the NHS Information Authority Portal Site. Use this site to gain access to other NHS Information Authority Websites using the Delivery Areas page or the drop-down menu above.

We aim to be the organisation that the NHS chooses to:

- Support national electronic care records to improve patient care
- Provide information services and knowledge for decision making
- Establish Health Informatics as a recognised national profession
- Provide reliable and secure information infrastructure services

The NHS Information Authority at HC2002 18-20 March, Harrogate

A close look at work leading to the development of a national Electronic Health Record was one of eight sessions run by the NHS Information Authority at 'Healthcare

Figure 5.3: Revealing menu items on a web-page based on available privileges

3. Having initialised the OASIS session state, the Portal sends a response back to the user. Since this response is generated on a per-user basis, it can be personalised appropriately for them. In the CBCL OASIS EHR prototype, certain links are added to a navigation menu depending on whether certain privileges were able to be granted to the user based on their role. This is shown in figure 5.3 – the left display represents the public view of a page, whereas on the right the privileged 'HRI Access for NHS patients' link has been revealed. The decisions which lead to a privilege being granted or denied occur within the Portal based on its policy file. In the case of our user, they must successfully satisfy the preconditions of the OASIS role activation rule to enter their parameterised 'patient' role on the Portal.

4. In our example we assume that the user requests to see their patient records via the customised link provided in the previous step. Were the user to be a health-care professional, this link would take them to a page from which they could select EHRs to examine for which they are directly responsible.

5. On seeing this request for EHRs, the NHS Portal contacts the Index Server. This communication is internal to the OASIS network, and occurs using SOAP over HTTPS. Again the HTTPS connection between these two nodes uses two-sided authentication; this time the server certificates of the Portal and the Index Server.

   The Portal provides the user token within whose session this request has been generated.

6. After successful connection establishment with the Portal, the Index Server creates its own OASIS session and associated token. Likewise, as for the Portal, the critical state for this session is written to a persistent back-end database. The structure of the Index Server sessions is almost identical to those of the Portal, since both use the same code base for session management (i.e. `OasisSessionServer` instances). Unlike the Portal sessions, however, the Index sessions also record a link field that contains the Portal token value.

   The Portal token value is maintained in order to facilitate the Index Server asking further questions of the Portal regarding the credentials of the user making the current request. This will be necessary when multiple credentials are required by the Index Service to grant a request on behalf of a Portal user.

7. The Index Server then retrieves the requested client records from its database. Divulging each individual record requires a successful privilege request to be made; that is, the prerequisites of the appropriate OASIS privilege authorisation rule must be met. The Index Server has its own OASIS policy, used to determine whether records will be released given the credentials of the requester.

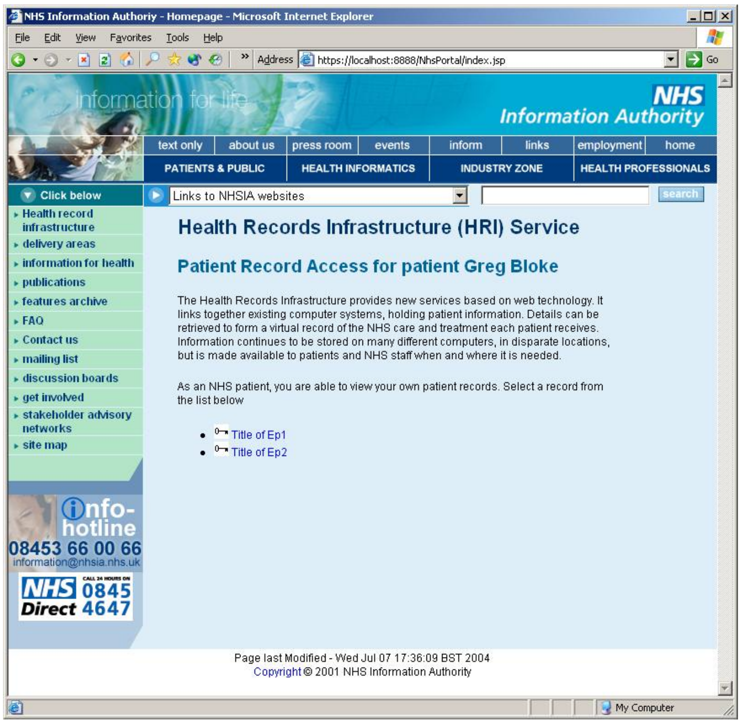
   Generally, Index Server policy will require the requester to be in some role in order for the 'Divulge EHR' privilege request to be successful. This enables the Index to perform a filtering function over the EHR record fragments returned for this request.

   Roles that are relevant only at the Index Server will be defined in the normal manner in the OASIS XML policy file. However, most of the enabling role membership information is actually at the Portal. Thus definitions are included in the Index Server policy file that mark roles as 'global', i.e. in this case they will need to have their state retrieved from the OASIS session at the NHS Portal.

8. As mentioned in the previous step, the Index Server will not initially know in what roles the user is active. This involves gathering required role membership information from the Portal site. This process of reverse communication is facilitated using the link token included within the Index Server session table.

   If a large number of records were being searched, performing network communication for each privilege request would rapidly become unnecessarily expensive. One approach is to issue Index Server role membership based on role memberships at the Portal. This approach requires explicit additions of such role definitions in the Index Server policy file. For convenience a credential caching strategy is employed even in cases where 'global' credential checks are being made – this strategy is described in section 5.1.3.

9. The Index Server's work in the EHR request is now complete. It returns the filtered list of EHR links to the NHS Portal site. Note that at this time there is next to no sensitive information in these EHR records, although future optimisations may store some further information on current medication and allergies for emergency



Health Records Infrastructure (HRI) Service

Patient Record Access for patient Greg Bloke

The Health Records Infrastructure provides new services based on web technology. It links together existing computer systems, holding patient information. Details can be retrieved to form a virtual record of the NHS care and treatment each patient receives. Information continues to be stored on many different computers, in disparate locations, but is made available to patients and NHS staff when and where it is needed.

As an NHS patient, you are able to view your own patient records. Select a record from the list below

- Title of Ep1
- Title of Ep2

Page last Modified - Wed Jul 07 17:36:09 BST 2004
Copyright © 2001 NHS Information Authority

Figure 5.4: The NHS EHR patient health record selection page

purposes. The Index Server never provides information beyond that required to contact the Health Care Organisations (HCOs) responsible for each EHR fragment. Indeed in our prototype even this information is proxied through the Portal, leaving users unaware from which particular HCO the Portal would potentially retrieve this EHR fragment.

Our design originally proposed an Index Server that merely returned URIs uniquely locating each record, although for convenience a document title field was added in our EHR prototype.

We do not preclude the Index Server having access to extra databases of policy-relevant material and using these to make its filtering decisions. These databases could facilitate fine-grained policy such as allowing patients control over which individuals have access or are denied access to their individual EHRs. For example, patients might be able to explicitly block certain records from browsing by health-care professionals even if those professionals would normally expect to be able to see all the patient's records. In this manner OASIS environmental predicates could interface with a confidentiality control model such as the TCM discussed above.

10. The Portal receives the list of EHR fragments returned by the Index Server, and formats them for the user (see figure 5.4). The CBCL OASIS prototype presents the records as a list of hyperlinks, each indicating nothing other than their record title. This formatting would probably be sufficient for patient queries, where even

over their whole life the number of EHRs is likely to be manageable on a single long page. For health-care professionals responsible for generating many patient EHRs per hour for which they will usually remain responsible, more comprehensive filtering and searching facilities would be necessary.

Each of the hyperlinks to EHRs are directed back to the Portal, rather than contacting the HCOs directly. Proxying HCO requests through the Portal aims to avoid undue extra resource requirements at each HCO site. For many HCO sites, particularly those using legacy software, it will be more convenient for them to send data in raw, data-oriented messages and rely on the NHS Portal sites to perform final data conversion and user-friendly formatting.

We assume that the user selects an EHR from the list presented to them, thus signalling to the Portal that the corresponding record should be retrieved.

11. Given that the NHS Portal still has the list of EHRs returned by the Index Service within this session, the Portal knows which HCO to contact in order to retrieve the EHR data.

    The NHS Portal opens a connection to the appropriate HCO Server. Again this connection is an internal OASIS communication, and is done using SOAP messages over an HTTPS channel. Two-sided authentication is performed using the Portal and the HCO server certificates.

12. Now the HCO Server creates an OASIS session. As for the Index Server, this session's critical data is stored in a back-end database, and it sets its link field to match the Portal's OASIS token value.

    The HCO Server also has its own independent OASIS policy file. Unlike the Portal and the Index Server, the policy evaluation engine at the HCO Server is in a position to make environmental predicate calls whose results are directly related to the contents of the EHR data.

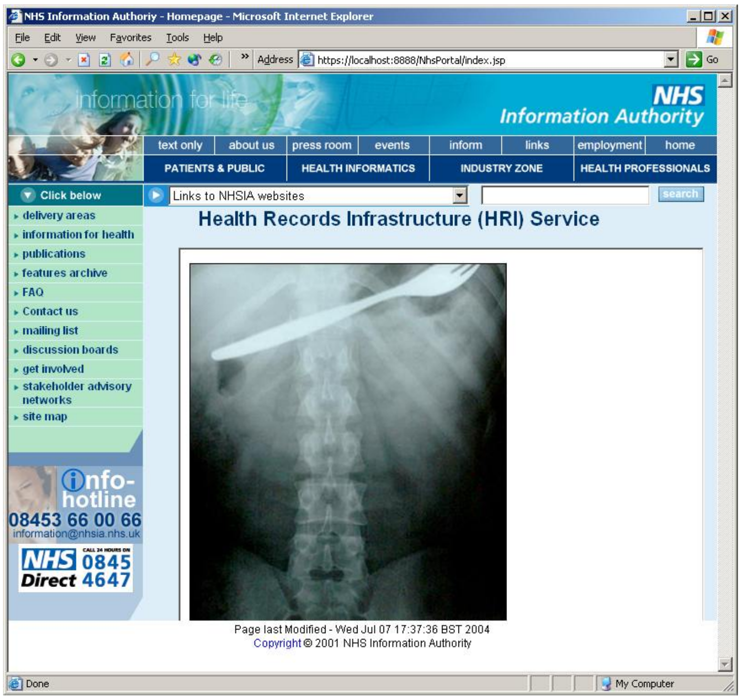
    If this HCO site is running legacy software, it will be the responsibility of the OASIS-aware software to translate between the Portal's credential structure and privilege requests, and those relevant to the underlying legacy EHR application.

    The HCO Server is now ready to retrieve details about the requesting user's credentials.

13. The HCO Server will make calls back to the NHS portal to retrieve role state. This uses the same mechanism as that for the Index Server, which was described above.

    When the HCO Server has all the credential information it needs to perform policy evaluation, it will be able to determine whether or not to grant the requests made by the Portal for retrieval of EHR data.

14. The work of the HCO server is now done. If the user's privilege requests were successful, it will retrieve the underlying EHR data. It is assumed that the HCO Servers will be able to package their responses into some form of XML communication



Health Records Infrastructure (HRI) Service

Page last Modified - Wed Jul 07 17:37:36 BST 2004
Copyright © 2001 NHS Information Authority

Figure 5.5: An example patient health record page

to be sent using SOAP over HTTPS back to the NHS Portal site. In our deployment of CBCL OASIS, we used standard web-based Multimedia Internet Mail Extensions (MIME) protocols, for example using XHTML to convey formatted text, and Portable Network Graphics (PNG) or other image formats to provide medical imaging data (see figure 5.5).

15. Finally the actual EHR data reaches the NHS Portal. If the EHR is delivered in a data-oriented XML message, the NHS Portal can apply transformations to make it more readable to the user. In our case XSLT would generally be the most convenient method for formatting XML data, since these services are provided by the J2EE application-server environment.

    Although we did not investigate this avenue of research, for some types of EHR queries it may be possible for the Portal to perform data aggregation at this stage before providing the final formatted EHR page to the user's browser.

16. At last the EHR data reaches the user's browser. If a multi-window approach was being employed and multiple EHRs had been requested, each EHR will have been laid out in its own block of space (e.g. using tables or frames). In such cases it is assumed that the human user will be able to see correlations between each data block, in preference to trying to program a computer to do so.

Although there are a large number of steps in the above protocol, many parts of the process can be performed in parallel. The abstractions leading to this parallelism are intended to increase the architecture's scalability. At the system level, the J2EE platform provides for server load-balancing, thus more computing infrastructure can be deployed to reduce bottlenecks in the system – particularly replication of the Portal, Index and HCO site servers.

5.1.3 OASIS extensions

This section discusses some of the extra requirements that were uncovered when OASIS was applied to the EHR prototype. In particular we examine integration of privacy protection, the use of a communication cost metric and policy visualisation.

Privacy protection

One of the benefits of using OASIS tokens to identify particular user sessions is that some degree of privacy can be provided. For a start, the OASIS tokens provide pseudonyms for the duration of a particular session. It is worth pointing out that the nature of EHRs will eventually make it reasonably obvious who the requests are coming from since the Index or an HCO Server will need to know the NHSID numbers of the records they are managing. Even so, it is possible that a number of role activations need to be performed before privileges are requested. During those role activations it is not necessarily the case that a principal will need to disclose their identity. They may instead be able to present capabilities issued to them that do not encode their personal details.

In the trace described in section 5.1.2, patient privacy is protected up until the stage where the Index or the HCO Server request role-activation state from the Portal. Local policy at the portal site could prevent such credentials being disclosed if that was the preference of the user.

A communication cost metric

As discussed in section 5.1.2, there are times when credentials need to be communicated between sites in order to perform policy evaluation. The CBCL OASIS implementation employs a scheme based on a cost metric to select the most efficient method to evaluate policy. The 'cost' has meaning in terms of computation and network time, but no explicit scale – it merely provides a mechanism for ordering evaluation strategies. At the moment the three main policy evaluation methods are:

Facts. The 'cheapest' evaluation comes from a credential being recorded as a 'fact'. Note that these facts are not actually axiomatic truths, but their validity is assumed to be very long lived. Facts might be generated in cases where it is known that the propagation delay of revocation information about a credential is longer than the current maximum session duration (so, a 'fact' within this session).

Rule. Rule computation is more expensive than merely looking up facts in the local fact database. It involves computing the validity of all the prerequisites of a given rule.

The actual quoted cost of a rule evaluation will be the sum of the costs of evaluating each of its prerequisites. In the cases where a prerequisite role has not yet been activated, it may be necessary to cost out the rules through which that prerequisite itself may be activated. This is similar to the dependency estimation approach we introduced in section 4.3.3.

Remote credentials. These credentials involve performing network communication to check the validity of a credential at a remote site. It is assumed that network communication is a high-cost operation, but as indicated in the user trace described in section 5.1.2, some inter-site communication will usually be required to start off local policy inference decisions.

When trying to activate a given role, or acquire some privilege, there may be a large number of policy rules that lead to it. One of the most important aspects of the cost metric concept is that it allows potential rules evaluations to be ordered. Thus a rule that activates a role based on credentials already active at a given site will be chosen over one that would need to request credentials to be sent over the network.

An interesting avenue for further investigation is how costs might change over time. Note that dynamic cost functions were not investigated in the CBCL OASIS implementation – the deployment environment was not sufficiently dynamic to have given us useful data for analysis. They resonate with some of the notions connected to trust and risk determination (see §6.8), but were considered to be beyond the scope of this thesis.

One useful dynamic cost might relate to the ‘freshness’ of credentials considered to be ‘facts’. The current implementation considers facts to remain true for the duration of a given user session, and gives them a very low cost (since they can simply be looked up in a table). Gradually raising the cost of using these facts over time would eventually lead to the ‘fact’ cost being higher than the ‘remote credentials’ cost. At that point the ‘fact’ would be refreshed from an authoritative source, and returned to its low cost.

Policy visualisation

We believe many users will more easily understand visual representations of policy than textual forms such as the current CBCL OASIS XML language.

Tim Mills of CBCL had experimented with using the AT&T Research Lab’s open source ‘DOT’ software [GN00, EGK$^+$02], to convert CBCL OASIS XML policy into the Scalable Vector Graphics (SVG) [W3C01a] format (see figure 5.6). SVG has two particular advantages over many other vector graphics formats. First, it is designed for the web and has wide-spread support in modern web-browsers. Second, it allows limited interactivity; SVG nodes can act as hyperlinks. When clicked they can perform actions such as redirecting the user’s browser to other URLs, or updating the contents of other HTML frames in a given frame-set.

The visualisation in figure 5.6 shows which OASIS prerequisites are on paths that reach a given privilege. In this case the privilege is `read_ehr` (the green diamond). Appointment certificates are shown as yellow ellipses, roles as red rectangles, and environmental

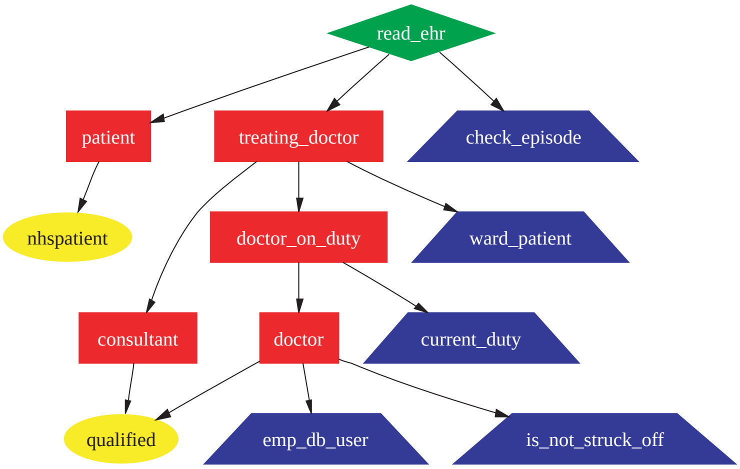


Figure 5.6: An SVG visualisation of OASIS prerequisite dependencies

constraints as blue trapezia. This figure offers no insight into how any given rule path will reach its target, however.

We developed the more comprehensive visualisation scheme presented in figure 5.7. Our scheme shows the individual paths from prerequisites to target roles (or privileges). The simplest example on the far left comes directly from the XML definition shown in figure 5.8.

As described in the key for figure 5.7, we show prerequisite conditions (yellow rectangles), with solid lines pointing to further conditions or target roles (blue ellipses) depending on them. In addition, dotted arrows point from bound prerequisite parameters (red diamonds), through named rule variables (white ellipses), to the unbound parameters of subsequent conditions or target credentials (green diamonds).

In addition to this static display, we modified the Apache Java-based 'Batik' SVG viewer [ASFc] to dynamically update the colours of each rule prerequisite. Using the Audit channel we added into the core CBCL OASIS implementation, we were able to gain a graphical display of rule evaluation progress. We envisage users having such a tool to help them determine why their role activation or privilege authorisation requests are not evaluating as they expect (it is a commonly held view that certain relationships are much easier to recognise when presented graphically than when embedded in text – see [BKR+98] for example). Our Batik application indicates which conditions are evaluating to true or false, and which conditions are not being evaluated at all.

5.1.4 Discussion

This section presents our CBCL OASIS electronic health record prototype for the NHS. We described the different components of our architecture, and provided an example interaction in which a given patient uses a web browser to securely retrieve some of their own EHR records. Overall, the CBCL OASIS prototype only scratches the surface of the

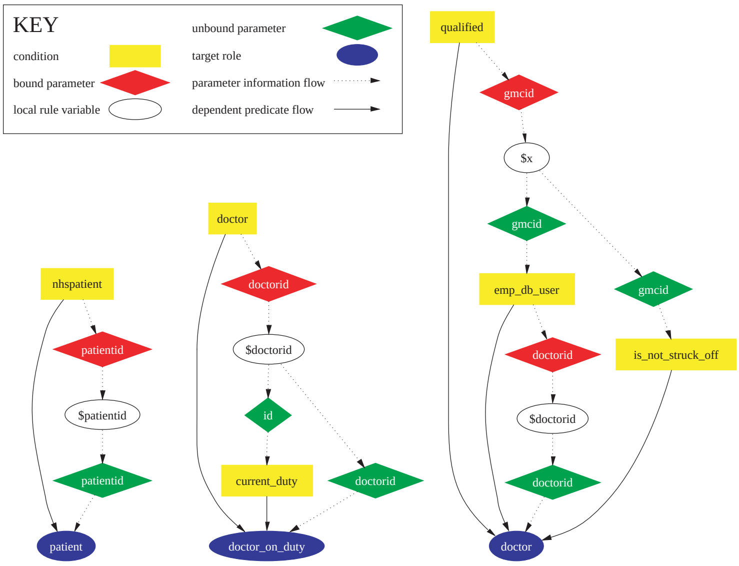


Figure 5.7: Visualising the sequence of OASIS rule evaluation steps

```
<activation role="patient">
  <condition name="nhspatient">
     <match name="patientid"    value="$patientid" />
  </condition>
</activation>
```

Figure 5.8: A simple OASIS XML policy rule

work required to implement the NHS EHR strategy. There is a great deal of further research and development that needs to be done, but our implementation demonstrates a basic network design and role-based security infrastructure on which an NHS EHR system could be built. Our use cases require the extra expressiveness of OASIS over many other RBAC implementations to support parameterisation, environmental interaction, and appointment.

The CBCL OASIS implementation led to a number of extensions to the original OASIS architecture needed to make it work in a web-oriented environment, and to more efficiently handle communication of credentials between OASIS services. We also experimented with some potential policy visualisation interfaces.

The CBCL OASIS EHR prototype has provided many insights into the security policy management and implementation challenges that still face researchers in the development of a comprehensive nation-wide EHR system for the NHS. In a nation-wide deployment, tools would need to be developed to manage versioning and deployment of policy files to the various network components and to ensure that potential policy conflicts were handled appropriately. It is likely that some degree of database interaction would be required in the access control rule evaluations (for example to check a principal was not on a black-list). These environmental predicates need careful attention to ensure that they do not violate the termination requirements of rule evaluation in OASIS.

Further interfaces would need to be designed to allow all users to set the access control policies pertaining to their own records. Many of the other trials of EHR systems have taken a higher-level view of the problem and focused on this user perspective.

5.2 Access control for distributed courseware

This section describes the design and implementation of a distributed courseware system using OASIS. The term *courseware* refers to education-related portal software. Generally courseware will allow students to receive and interact with educational units within some course structure. It is a good environment for testing parameterised RBAC systems, since the number of functional roles is rather small, but there can be many case-by-case exceptions. For example, a student in one course may be an instructor in another – forming the cross product of all courses with all roles is an administratively inconvenient mechanism for handling such cases versus the ease of using role parameters.

There are many other mainstream courseware infrastructures, e.g. Blackboard [Bla98] and WebCT [Web97], that focus on the design of portal web-sites. Our particular interest was how we could support access control for institutions collaborating with each other in the delivery of their educational units.

5.2.1 The RAED project

Research in Role-based Access-control for Evolution of Distributed courseware (the RAED project) [NLN$^+$04] was initiated by Strathclyde and Glasgow Caledonian Universities. We provided the OASIS implementation and re-deployed it to fit the requirements of the project. The OASIS model was chosen because the 'tertiary courseware' (described below) at the heart of the RAED project must make per user and historical access control decisions. It was desirable to use an access control environment that was distributed, had a highly expressive policy language, and allowed the integration of database queries into policy inferencing.

5.2.2 Background to the RAED project

Before describing where the OASIS access control system fits into the RAED project, we provide some research background and terminology used within the project.

Vicarious learning

The pedagogical basis for the RAED project was first investigated in the MANTCHI (use of Metropolitan Area Networks for Teaching Computer Human Interaction) project – a Scottish Higher Education Funding Council supported collaboration between Glasgow Caledonian University, the University of Glasgow, and both Heriot-Watt and Napier Universities in Edinburgh. The MANTCHI project used tertiary courseware in the context of human-computer interaction exercises. Being a design-based discipline, such exercises were proposed to be suitable for work-based learning. Students can benefit from selective access to previous student-student and student-tutor interactions, allowing the "learner's voice" to be heard [NLE+04].

Tertiary courseware

The RAED project is designed to implement a system that administers 'tertiary courseware'. Mayes [MN99] provides a three stage model of learning; conceptualisation, construction and dialogue. These stages map onto primary, secondary and tertiary learning technology, as described below:

Primary courseware. Primary courseware is usually considered to be material provided directly by the teacher of a particular course. As a consequence it will most likely present core subject material. A current problem is that the content experts are not usually also courseware experts. Development of courseware units can thus become a highly expensive and time consuming process (as explained by Leeder et al [LWS+04]). This is compounded by a lack of software designed for easy manipulation of courseware units.

Secondary courseware. The environment in which the learner is actually going to perform learning tasks, and the task materials themselves, are described as secondary courseware. There is no widely deployed solution for this type of software, largely because the most efficient pedagogical practice has not yet become apparent (most projects are gradual transitions from existing teaching practice). Secondary courseware facilitates interaction between students and subject experts placed within the environment. Web-based discussion boards are emerging as a common implementation of such learning environments.

Tertiary courseware. Tertiary courseware describes an environment in which students are able to learn from snapshots of previous secondary courseware interactions – in other words it facilitates learning through the observation of past learners learning. The dialogues captured may also involve tutors or assessment outcomes being posted, as well as discussion among students.

In the case of the RAED project, the implementation provides an environment that modifies web-pages based on the authorisations of the principals (i.e. learners, teachers and administrators) it is able to authenticate. For example, students might be provided access to the contents of previous discussion boards if this did not violate policy formed on the basis of the educational atoms and trails (described below).

Facilitating the use of tertiary courseware requires careful thought about the content of the primary and secondary courseware to which it is connected. A particular type of problem needs to be tested rather than a specific exercise. In all cases the skills needed to solve such problems should be presented in the primary courseware. There needs to be a series of related secondary courseware units – learners use tertiary courseware to enhance their interaction with this secondary courseware. Of course users must not be given access to tertiary courseware too closely related to the exercise currently being used to assess them.

Atoms and trails

The relationship between atoms and trails is indicated in figure 5.9. Atoms are units of secondary courseware that provide a problem-based learning task. Trails are tertiary courseware built up from student solutions, student-tutor discussion, and student-student discussion.

A trail will be provided to students for the sake of them completing a given atom. Of course trails represent the material that leads to the solution of an earlier exercise within that atom. Thus, new exercises need to be generated for each set of students. We talk of atoms having some number of instantiations. Even so, each atom will have some invariant parts, that will usually be related to the primary courseware for which this atom's exercises have been set. For example, MANTCHI looked at interface modelling using state-charts, where given instantiations might relate to designing a Walkman interface, or the interface for a radio alarm, and so on.

5.2.3 Where OASIS fits within RAED

The overall structure of the RAED infrastructure is shown in figure 5.10. OASIS satisfies a number of functions within RAED. Firstly, it provides a scheme for the authentication of

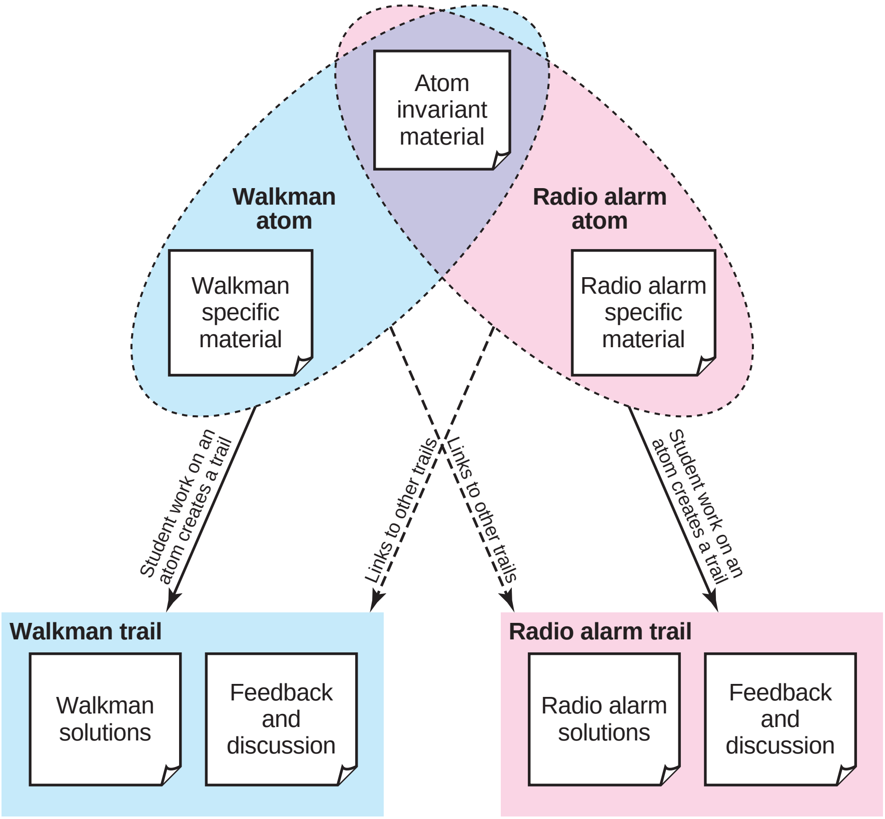


Figure 5.9: Atoms and trails

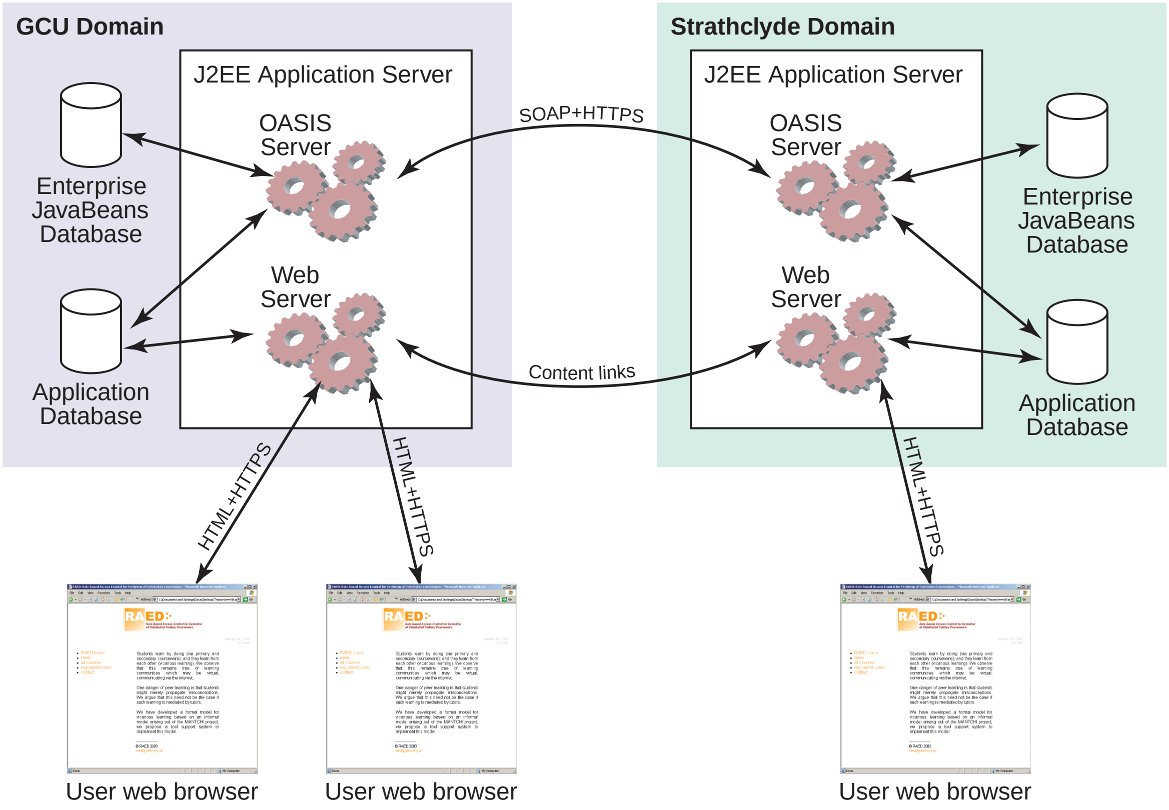


Figure 5.10: The RAED project infrastructure

users. The X.509 certificates used for authentication are similar in structure to those used in the EHR prototype, although naturally we use a RAED-specific set of OASIS attributes and appointments.

As well as providing authentication services, OASIS works with the J2EE application server to customise web-page content based on the privileges a principal may obtain. This policy is encoded in the CBCL OASIS XML policy format. In particular, environmental constraints are used to enforce access control based on history – for example, checking that the students working on a particular atom exercise may access a trail only if it does not contain solutions to their current exercise.

There are four main categories of RAED authentication certificate; students, subject experts, tutors and local organisers. Each of these categories is described in table 5.1.

The privileges in the RAED system are read, create and update permissions for different types of web content relating to atoms and trails (see table 5.2).

The relationship between users, and membership of the atoms and trails, needs to be updated after each exercise. We used OASIS environmental predicates that perform external database interaction to evaluate user privileges, rather than coding these relationships into the policy files themselves. The next section explores how the RAED application has led us to develop a more general extension of database support for parameterised RBAC

Certificate type	**Description**
Student	A student will be denied access to trails that are directly relevant to their current atom instance. They will be denied access to most other RAED pages, since these contain tutor-oriented information, and administrative controls.
Subject expert	An atom expert is the author of a piece of secondary courseware. They will have administrative privileges to update that atom.
Tutor	Tutors are given the privilege to view all of the trails for a particular atom. They are also given access to teacher-oriented information submitted by the subject expert. Note that tutors do not need to be at the same institution as their students.
Local organiser	Local organisers have the administrative privileges to assign students to particular instances of atoms, and regulate the tertiary courseware to which these students have access. Local organisers will set deadlines and make institution-specific modifications to the atoms seen by the students in that institution.

Table 5.1: The four main categories of RAED authentication certificate

Privilege	**Description**
atom-core	These privileges provide access to the background material for a subject. The subject-expert may submit details such as a reading list, or other supporting material.
atom-ex(n)	These privileges protects content written by the subject-expert – this time material relating to each particular exercise in an atom, here shown as a parameterisation in the exercise number, n.
trail(n)	The trail privilege category protects the tertiary material for the current exercise. It has the current exercise number as a parameter (n) so that access can be blocked to previous solutions to that particular exercise.
atom-local(d)	This set of privileges protects local configurations for a particular atom, e.g. specific assignment due-dates.
atom-notes(n)	These privileges allow access to teaching notes and other resources useful to tutors for exercise n, e.g. past exam questions and answers. All roles can access these notes, except for students, whose access will be restricted by local policy. For example, some organisations may allow students to see solutions soon after assignment deadlines have passed, whereas others may choose to leave all such material restricted.

Table 5.2: The five main RAED privilege categories

systems.

5.2.4 Database support for parameterised RBAC

During the design of the RAED project, it became clear that there was a frequent need to access database tables external to the OASIS policy files. This was necessary to avoid writing policy rules of too inconveniently-fine granularity (i.e. relating to individuals). For example, when designing rules for tutors' privileges, we need to handle the situation in which a tutor on one course might actually be a student in another.

One potential solution to this sort of exception case is to add support for negative permissions into the access control system. Whilst this increases the expressiveness of security policy, it also complicates the inferencing process. In our OASIS research, we have usually found that it is possible to avoid requiring negative permissions – instead environmental checks are added as guards on the appropriate policy rules. In the RAED case, we used existing student administration databases to narrow privilege checks down to a per-user basis.

The presence or absence of matching records within external databases provides a multi-level policy description. The overall policy rule 'templates' are defined in the OASIS XML policy files. The fine-grained permissions are then defined in the database tables referenced from the template rules. The next section explores tightly integrating database access into parameterised RBAC policy.

Integrating database access into parameterised RBAC policy

On the basis of our RAED experience, we suggest that a facility for database checks should be a first class member of parameterised RBAC policy languages. In our case, the J2EE application server environment on which CBCL OASIS operates requires an SQL back-end database for the persistence of its Enterprise JavaBeans. Thus if database lookups are integrated into an OASIS-like policy language, they can take advantage of this existing database infrastructure. In terms of management, higher-level policy language constructs will allow clearer visualisation and parameter modification – it avoids undue 'programming' within OASIS policy rules.

Although our specific examples discuss constructs that relate to the OASIS policy language, this section has much more general applicability to parameterised RBAC systems – we argue that the predicate-like semantics of particular database operations justify them being integrated into policy languages more tightly than other external interactions.

A relational database table is itself a predicate – a view well put by Chris Date in his key-note talk at the Very Large Database Conference (VLDB) in 2002 [VLD02]. Truth of a particular predicate expression will depend on whether a row exists in some given table.

Consider how a database lookup might be performed using OASIS environmental predicates. For the moment, we consider the simpler example of database lookups where we do not retrieve any parameter values from the database – i.e. all parameters to the environmental predicate are already bound. Two main approaches exist; in the simpler approach, environmental predicates are hard-coded to perform particular database operations on particular tables. Whilst straightforward, this approach causes the database connection to be implicit from the perspective of the policy file. In particular, the definition section of the policy file does not distinguish between this sort of database query and any other environmental predicate.

A slightly more complex approach is to provide an environmental predicate that performs general database lookups. It must contain a parameter indicating which table to access, and uses some heuristics to determine which parameters correspond to related database field values. This approach has the advantage that it does not need a separate environmental predicate defined for every type of database lookup performed. Because of this, it was the approach we used in the early RAED implementation work. However it is dangerously close to programming policy as if it was a general-purpose programming language – database-specific details have to be coded into each rule. This may cause errors to emerge given there is no coupling between rules and the database-specific properties.

A cleaner approach promotes such database checks to first-class members of the policy language. Consider the following role activation rule taken from the RAED project:

```
<activation role="student_on_atom">
  <condition name="student">
    <match name="id"     value="$x" />
    <match name="atomid" value="$y" />
  </condition>
  <condition name="DBCheck2">
    <with name="table"  value="class_member"/>
    <with name="field1" value="raed_id" />
    <with name="value1" value="$x" />
    <with name="field2" value="atom_id" />
    <with name="value2" value="$y" />
  </condition>
</activation>
```

The condition DBCheck2 is a customised environmental predicate which translates requests into SQL statements run against the J2EE back-end database. Its definition in the policy file is:

```
<environment name="DBCheck2" class="raed.environments.DBCheck">
  <input name="table"  type="java.lang.String" />
  <input name="field1" type="java.lang.String" />
  <input name="value1" type="java.lang.String" />
  <input name="field2" type="java.lang.String" />
  <input name="value2" type="java.lang.String" />
</environment>
```

If we raise database lookups to first-class policy language members, we simplify the role activation rule:

```
<activation role="student_on_atom">
  <condition name="student">
    <match name="id"     value="$x" />
    <match name="atomid" value="$y" />
  </condition>
  <condition name="CheckStudentDB">
    <with field="raed_id" value="$x" />
    <with field="atom_id" value="$y" />
  </condition>
</activation>
```

The definition of the `CheckStudentDB` condition is significantly simplified:

```
<dbcheck name="CheckStudentDB" table="class_member" />
```

In the RAED project, we were able to effect this language simplification with an XSLT preprocessing transformation prior to deployment of the CBCL OASIS policy files.

Binding output parameters

Relational databases are clearly appropriate for determining set membership. However, they are also a convenient mechanism for type transformations in access control policy. For example, a given policy specification may predominantly use ID parameters local to a particular site, but use IDs with a global scope for initial authentication. A database stored at the local site would provide a means for authenticating with the global ID, but activating all user sessions using local ID parameter values. This requires the database interaction predicates to assign values to policy rule variables.

More formally, let us consider an environmental predicate **extDb** performing database interaction with the signature:

$$\mathbf{extDb}(bound_1, bound_2, ...bound_{nb}, unbound_1, unbound_2, ...unbound_{nub})$$

All of the $bound_i$ and $unbound_j$ are parameters in an OASIS rule. In the case of a simple predicate check (e.g. as used in the RAED database interactions), nub is equal to zero. The database effect can be likened to evaluating true if the number of rows found executing the following SQL is greater than zero (we set nb at two for this example):

```
SELECT * FROM table WHERE f1=bound_1 AND f2=bound_2;
```

Consider instead the effect of a conversion function between IDs, where nb and nub are both equal to one (as an example without loss of generality to higher nb and nub values). Again the database effect expressed in SQL would be similar to:

```
SELECT f2 AS unbound_1 FROM table WHERE f1=bound_1;
```

We have three particular possibilities for the cardinality of row results:

No rows returned. In this case, the policy mapping request has failed, and the environmental predicate evaluates to false. Note that this situation might arise due to other software or hardware failures instead of reflecting the true state of the database tables.

One row returned. This is the most desirable case. We have a simple mapping, the predicate can bind the hitherto unbound variables and the rule inferencing process may continue.

Many rows returned. In this situation we have a choice of behaviour. The OASIS rule evaluation process, being designed for predictable termination properties, does not have the capacity to back-track through alternatives when later steps of the policy rule evaluation fail. In such cases, we would need to 'dumb down' the result to a single binding – the simplest option being to take the first row that matches as being the definitive answer.

In order to maintain fast policy evaluation, it is preferable to avoid multiple bindings in database lockup predicates (the third case in the above list). If the primary key of the relation being queried is some subset of $\{bound_1, bound_2, \ldots bound_{nb}\}$, we guarantee that the multiple-binding case will not occur.

5.2.5 Discussion

One particular aspect of future work in the RAED project relates to inter-organisational privilege authorisation. At the moment, presentation of RAED appointment certificates allows cross-institutional authentication – students from one organisation (say Strathclyde) may thus be given access to resources at another (say Glasgow Caledonian University – GCU). However, at the moment GCU would need entries in its policy file relating to these Strathclyde students.

The CBCL OASIS infrastructure potentially allows a much more powerful form of inter-organisational cooperation through its inter-service SOAP interface. If a Strathclyde student were to request a GCU privilege that needed a particular student role, the GCU OASIS service could check that *Strathclyde* is able to activate the required role for this student, rather than having to do so itself. This then allows institutional collaboration to occur on the basis of roles alone, avoiding the need to share student information records for initial authorisations.

In terms of our implementation, some subtle errors were fixed relating to the CBCL OASIS code-base storing duplicate logging information. Also, a facility for a separate audit stream was added, although it is not yet used in the RAED project.

In future we hope to use OASIS as an access control monitor for 'DSpace' archives [TBB^{+}04]. The University of Cambridge is at present testing DSpace technology – it is thus likely to be used for storing the course units behind educational portal infrastructure currently being developed by CARET. We are keen to use our RAED experiences to assist CARET integrating the much more expressive OASIS policy language in place of the existing DSpace security technology.

5.3 Conclusion

In this chapter we present the NHS EHR and RAED case studies. Both projects use similar OASIS implementations, but demonstrate the architecture's applicability to very different domains.

We also present a number of OASIS extensions supporting required features of each application. The EHR project led to the development of cost metrics and visual presentation and analysis of OASIS policy. The RAED project led to improvements in the policy language needed to enable the specification of database restrictions on role activations and privilege authorisations.

As a benchmark, we evaluate OASIS within these applications against the principles presented in chapter 3. Both applications clearly implement distributed privilege checking architectures, although only the EHR application actually uses the inter-OASIS service SOAP protocol to perform dynamic role activation requests.

Both applications make use of the OASIS environmental interaction support. The RAED project made significant use of environmental predicates to integrate database queries into role activation rules. This also provides a means to implement grouping of sets of principals.

Because both applications use the OASIS policy structure, the evaluation of any given policy rule is guaranteed to terminate, provided that any environmental predicates called do so also (although this is not explicitly enforced). It is also possible to modify policy rules while the system is in operation, and have these policy changes reflected immediately.

In many other respects CBCL OASIS falls short – particularly in terms of dynamic credentials and constraints. Neither application uses fast credential revocation or notification-based dynamic constraints, yet in both cases such facilities would improve security. The extensive reliance on database querying in the RAED project within role activation rules does increase the degree to which RAED policy can be rapidly modified on a per-user basis, however. The SOAP inter-service messaging framework could facilitate fast revocation of credentials, but does not currently do so. Also, the OASIS policy language provides no explicit support for dynamic constraints.

Finally, the administration and audit management of CBCL OASIS is impracticably complex. Auditing is achieved through a log file, but no concerted efforts have been made to allow easy analysis of the policy computation; this log also includes a great deal of system-level diagnostic output. Policy is accessed at the granularity of a single XML file, and does not provide a framework for self-administration. Whilst external multi-level policy management could be applied to this file, no particular software has been incorporated to do so. It is worth noting that CBCL OASIS policy *is* dynamic in the sense that updates to the policy file are immediately reflected in the CBCL OASIS state, however.

Overall, both the CBCL OASIS EHR prototype and RAED applications have been extremely useful tests of OASIS concepts. They have highlighted areas of OASIS that need further development to meet the needs of a comprehensive distributed privilege management system. Many of these developments have been addressed in the EDSAC architecture which is introduced in the next chapter.

6 EDSAC21

This chapter introduces our model for Event-driven Distributed Scalable Authorisation Control (EDSAC21). This model builds on OASIS by introducing publish/subscribe event-based communication to allow distributed nodes to participate in access control decisions.

This chapter first describes the motivation behind the design of EDSAC (§6.1), and how it relates to OASIS. We then introduce the four layers of the EDSAC architecture. Section 6.2 discusses the event types needed to support the security functions of the EDSAC architecture. Next, we discuss the individual components of EDSAC nodes, and how these nodes are linked together (§6.3). Some of the potential dynamic distributed access control features facilitated by the EDSAC approach are listed in section 6.4.

Because we need a publish/subscribe event delivery infrastructure to facilitate synchronisation of EDSAC state, as a side-effect we are able to provide applications with a policy-controlled interface to this event dissemination framework (§6.5).

Section 6.6 discusses the formation of hierarchical EDSAC federations and considerations necessary for establishing the EDSAC network synchronisation period. We then present recent research we have done building encryption-based security into the publish/subscribe paradigm in section 6.7. Finally, section 6.8 briefly discusses trust-based replacements for identity-based authentication.

6.1 Introduction

The acronym EDSAC has already been used within the Cambridge Computer Laboratory: it describes the Electronic Delay Storage Automatic Calculator, an early digital computer first operational in 1949. For this reason, we append 21 to indicate the 21st century, although we shall often refer to the architecture just as EDSAC since the context is clear within this thesis.

Much of the motivation of our work is encoded into the EDSAC acronym – Event-driven Distributed Scalable Authorisation Control. We emphasise the dynamic nature of our security architecture by it being ‘event-driven’. That is to say, the status of credentials – and through them – privileges, can be updated at high speed using a dedicated communication infrastructure.

The use of ‘distributed’ and ‘scalable’ indicates our objective: to undertake a design for a truly Internet-scale security architecture. This includes the need to provide scalable systems for policy management, and principal status dissemination within any implementation.

Finally, our use of the term ‘authorisation control’ as opposed to ‘access control’ emphasises that security architectures need to be able to react to changes in the credentials of their principals at any time, and not merely perform a restrictive function at the time of a given principal’s access requests.

We emphasise that our proposal is for an architecture, rather than a particular implementation. Whilst we provide an OASIS-like policy language for our prototype, we do not want to tie down a particular method of policy rule computation. We instead want to capture the common communication requirements of all role-aware access control methodologies. Our approach aims to unify the different rule evaluation engines that may suit each part of a distributed access control application. A desirable side-effect would be to allow side-by-side comparison of the different schemes currently being researched. We propose an architecture more expressive than either ISO/IEC 10181-3 or the American National Institute of Science and Technology (NIST) RBAC standards, since we feel this is necessary to capture salient aspects of a wide range of current RBAC applications and research projects.

6.1.1 How EDSAC relates to OASIS

Given that this thesis covers a significant amount of OASIS research, it is important to indicate how the two systems relate to each other.

OASIS is an access control architecture that provides a particular policy language, a rule evaluation engine, the concept of persistent appointment credentials, and the requirement to support revocation through event channels. The EDSAC architecture is more general than this. We decouple the policy language from the protocol used to communicate changes in credential state between different nodes in an access control network. We could potentially support different rule evaluation semantics on different nodes, provided that they were all ‘role aware’ and supported credential activation and revocation.

Because of our experiences in deploying OASIS, our current EDSAC prototype uses the OASIS policy language extended with additional syntax (where necessary) to support new EDSAC features. When we refer to the EDSAC policy language, we mean our extended OASIS language. The context should make this distinction clear.

The EDSAC architecture contains many features that do not relate to any OASIS notion. Examples include two-phase role activation, deferred events, and dynamic policy definition. These features are discussed in the sections below.

6.1.2 EDSAC layers

The EDSAC architecture consists of four primary design layers. Each individual EDSAC node protects its own Application Programming Interface (API) using layer 0 logic. The node cannot rely on anything other than knowledge in its session database and local environmental calls being available to determine whether to grant a privilege to a requester. Layer 0 also protects the publish/subscribe system interface. An EDSAC network is formed by EDSAC nodes communicating with each other using layer 1 logic. At this layer, rule inference can use layer 0 privileges to retrieve credentials from other EDSAC nodes via events.

Naming of policy elements must be unique within any EDSAC network. In our prototype we did not experiment with EDSAC federations and thus were able to use unscoped policy element names. Although we have not focused on the naming of policy elements in this thesis, note that our appointment certificates do use a scalable, hierarchical naming scheme. This is because, as for the CBCL OASIS implementation, the X.509 certificates include an X.500 distinguished name.

An interconnected network of EDSAC nodes is insufficient to encode application semantics. Layer 2 logic defines the procedures EDSAC nodes use to achieve application functions on behalf of their authenticated principals. It is also the appropriate level at which to designate which nodes manage directory services, if this is necessary for any particular deployment. The descriptions of deployment-specific roles, rules, and credentials stored in these directories allow EDSAC nodes to search policy meta-information dynamically. This would allow policy authors to write new rules without having to interact personally with other policy authors. It might also be employed in future to allow an EDSAC node to adapt their credential requests to policy elements created after that node's deployment.

At layer 3 we build hierarchical federations of EDSAC networks. Federation of EDSAC networks is necessary when the size of a single EDSAC network becomes impractical. Given the cooperative nature of our role activation protocol, it is important that all interested parties have an opportunity to participate in policy decisions. The size of the network and the heartbeat period will together determine how frequently nodes lose synchronisation with policy events (note that such nodes will be able to detect they have lost synchronisation). Ideally, nodes should remain synchronised at all times. As a consequence, breaking large networks into hierarchical federations may be preferable to increasing the heartbeat period. Alternatively federations may be necessary when various EDSAC networks are operated by different organisations, and the links between these EDSAC networks require communication gateways.

Federations of EDSAC networks require mappings between the name-spaces used within each EDSAC network. It is of course possible that a number of EDSAC networks share the same name-space. Note that the research into the scalable naming of events is being explored within the Hermes system. We expect to integrate these developments directly into EDSAC policy name-spaces.

Section 6.2 examines the event-based communication by which layer 0 EDSAC services can achieve layer 1 functionality. Section 6.3 discusses the structure of EDSAC nodes (layer 0) and EDSAC networks (layer 1). We provide examples of layer 2 logic in section 6.4.

6.2 The EDSAC communication framework

This section discusses the core communication primitives required for an EDSAC implementation. For distributed applications in which immediate revocation is crucial, or in which the credentials and privileges of principals change frequently, we argue that the infrastructure for disseminating such updates should be tightly integrated with the access control system.

This belief has always been implicit in past OASIS publications (such as [BMY02a]), but the specifics of the interface to the event middleware have generally not been explored, nor thoroughly implemented in prior prototypes. EDSAC takes a significant step beyond the OASIS project by integrating secure publish/subscribe messaging rather than simpler event-based delivery. The extra power of such messaging infrastructures directly supports many of the EDSAC features discussed in section 6.4.

6.2.1 Communication layer requirements

We require that the underlying publish/subscribe communication layer performs ongoing connectivity testing between neighbouring event brokers. It is necessary that each EDSAC node knows when it should suspend local activities or revoke roles if it becomes clear it cannot currently receive push communications from other EDSAC nodes. Note that this does mean that if the publish/subscribe network itself is disrupted, it is possible for a now-unauthorised principal to carry out activities after a lost revocation event for up to the heartbeat period's duration.

It is also required that the communication layer does not fail silently when an event is not successfully delivered throughout the event dissemination spanning-tree. Failures of this type should be treated in the same way as losing heartbeat synchronisation.

Finally, we require that the communication layer maintain a monotonically increasing heartbeat counter, or 'network tick' to enable an overall temporal ordering of events (the latter term was introduced to describe the heartbeat counter in IBM's Gryphon publish/subscribe system [IBM01]).

From the EDSAC perspective, network ticks allow groups of events to be collected together in a manner agreed across all communicating nodes, or for nodes to establish they have lost synchronisation. We discuss how nodes should manage 'simultaneous' (with respect to tick number) EDSAC events and how to resynchronise security state in section 6.4.4.

We use network tick numbers to define the deadlines on votes taken by EDSAC nodes as to whether certain role activations should proceed (§6.2.2).

6.2.2 Event types

The eight event types core to layer 1 of the EDSAC architecture are shown in table 6.1. We also indicate the attributes required by each. The purpose of each of these types is discussed in the sections below.

Event type	**Source filter**	**Target filter**	**Pattern**	**Other attributes**
Credential request	yes	yes	credential	deadline
Credential forward	yes	yes	credential	
Credential revoke	yes	yes	credential	
Role activation request	yes	yes	credential	action time
Role activation vote	yes	yes	credential	vote weight
Role activation confirm	yes	yes	credential	
Action request	yes	yes	rule identifier	
Privilege vote	yes	yes	privilege identifier	vote weight

Table 6.1: EDSAC core event types and their attributes

Before discussing each of the core event types employed by the EDSAC architecture, we must explain some of the event attribute terms used in table 6.1. The source and target filter attributes allow narrowing of the scope of event delivery. Since we are using content-based event delivery, we do not need to specify explicit source or target addresses.

In OASIS, user sessions act as a 'container' within which a given principal's roles will be activated. When developing our EDSAC prototype, it became clear that we also needed containers in which to monitor dynamic constraints. Parameterised policy contexts (§4.4.2) were used to provide dynamic constraint containers. These two types of container are orthogonal: all active roles will belong to a user session, but they may also be stored in any number of contexts. From the perspective of the EDSAC architecture, however, both user sessions and policy contexts are features specific to our prototype. Both would be stored in the layer 0 session database.

As discussed in chapter 7, we found it useful to occasionally apply event delivery target filters based on policy contexts. Thus some events only reach nodes that have activated roles within some given context, rather than the entire EDSAC network.

Note that in our prototype the use of source filters was not explored. They have been provided to facilitate future schemes which support negotiated credential disclosure – credential stores would enter into multi-phase communication with the requester before choosing to satisfy their credential request.

If a particular node publishes events with a given source identifier and also subscribes to appropriate events filtering on this identifier, reply messages can be sent through the publish/subscribe system to the original publisher. It may be possible that the efficiency of event delivery can be increased if the underlying publish/subscribe system has a notion of reply messages and support for one-to-one communication and to-one and to-many event reply capabilities. We are working with the designer of Hermes (Dr Peter Pietzuch – see [Pie04]) to develop these ideas, and are hopeful that some will be included alongside ongoing research into secure Hermes extensions.

Another key concept is that of *action times.* This attribute is present on the role activation request event, and facilitates EDSAC two-phase role activation. When interested parties hear of a given node's intention to activate a role, this attribute provides a period of time in which conflicts can be declared and this role activation avoided. In our tests, we used a network heart-beat which contained a serial number (also known as a network *tick*. Similar time measures exist in publish/subscribe projects such as [IBM01]) to test for network synchronisation. If the synchronisation period is set as we discuss in section 6.6, four network ticks can be assumed sufficient for other nodes to have a chance to vote on this request. In our experiments, rejection votes are registered when dynamic constraints would be violated were the role activation to be allowed to proceed.

The deadline attribute of credential request events is also a network time measure, although in this case it merely indicates how thorough the requester wants the credential search to be. In a federation of EDSAC networks, the deadline might be used as an indication at gateway nodes that this credential request should not be forwarded to other domains.

Many of our events will not need source or target filters; instead they will route messages to EDSAC nodes interested in some set of credentials, rules or privileges. The 'Pattern' column in table 6.1 indicates which form of filtering is appropriate for each type of event.

Turning our attention briefly to implementation concerns, for this filtering to function efficiently, the publish/subscribe system must be able to interpret how any given attribute expression covers the instances of that particular attribute. In the case of the Hermes publish/subscribe system [PB02, Pie04] used for some of our testing, coverage relationships include inequalities on integer-valued attributes, and prefix matching on string-type attributes. Because our prototype was implemented in Prolog, we did not use a tight coupling between Hermes data types and our use of attributes – we treat all attributes as strings.

We also developed an internal Prolog publish/subscribe system emulator for local testing that (unsurprisingly) used Prolog unification to test whether a given instance of an event attribute fitted within an attribute pattern. This is more expressive than the Hermes system with respect to partial match queries on functional term components. Such queries can be integrated into Hermes attribute coverage testing through mangling of the attribute names to represent the elements of an attribute in functional form. A more elegant solution could be engineered into Hermes' attribute coverage testing if required. Note also that Hermes provides features we do not yet use, such as its ability to place events within a type hierarchy.

Credentials versus roles

The events presented in table 6.1 make a distinction between credentials and roles. Before we examine each of these events in detail, we should clarify our use of both terms.

In OASIS rules, appointment certificates are a form of credential. They persist beyond the duration of a given OASIS user session, and may well be understood by non-OASIS security technology (e.g. web browsers in the case of our CBCL OASIS X.509 certificates). Apart from environmental predicates, the only other type of prerequisites are roles. We argue that although they cannot be presented to non-OASIS technology, and they are only valid when active in a user session, they are still a form of credential.

The primary difference between roles and other credentials in EDSAC is that dynamic constraints may be triggered during the activation of a role. There is no real notion of the activation of a non-role credential – we assume that from the perspective of the local policy evaluation engine, its status is registered as changing from 'unknown' to 'true' or 'false' (if the published credential request event receives a reply).

Of course in terms of revocation, roles and other credentials are treated equivalently. In either case, if a revoked credential or role is a membership prerequisite for other credentials or roles, the target credential will also be revoked.

Credential request events

The credential request and credential forward events allow the policy decisions at a particular EDSAC node to be informed by credentials stored at other nodes within the EDSAC network (or other linked EDSAC networks).

The credential request event is used by a service that lacks information it needs about prerequisites in order to activate a role. To use OASIS inferencing as an example, if we skip locally-evaluated environmental predicates, there are two types of prerequisites. Credential requests for appointment certificates assure the requester that some service has a valid version of the named X.509 certificate in their possession. Credential requests for role membership indicate that the named role is active under the same user session somewhere in the EDSAC network.

The publishers of credential request events are the sites that will be evaluating policy. Event publications will be emitted as a consequence of policy rule evaluation.

To increase the performance of the underlying message delivery, advertisements of credential request events can occur before the need for actual event publications. In the CBCL OASIS case, for example, a credential request advertisement could be made as soon as a user session is created on a node which contains 'global' scoped preconditions (these are prerequisites that may be queried in a 'pull' manner from the connecting service – see chapter 5). The requesting node would need to have first subscribed to credential request events for these pull queries to be successful.

The subscribers to credential request events are sites which have active roles, or which maintain credential stores such as LDAP directories of X.509 certificates. The subscriptions made should be as narrow as possible, to increase the efficiency of event routing through the publish/subscribe system.

Credential forward events

Credential forward events are primarily intended for replies to credential request events. The subscribers and publishers are reversed in function from that already discussed for credential request events. We use the term 'forward' to indicate that a credential holder is forwarding a copy of the credential for which they vouch the current validity. Although the publish/subscribe system is used for revocations anyway, and thus we do not need to track dependency connections explicitly, nodes can choose to monitor the targets to which they forward credentials.

Were credential forward requests solely for the purpose of replying to credential requests, we could deliver them directly to the requesters without using the publish/subscribe system or requiring a separate event type. However there may be situations in which we want to forward a credential to multiple EDSAC nodes simultaneously, and thus would benefit from content-based event delivery.

There are two main reasons a credential forward event might be published. Firstly, credential stores may automatically publish such events in response to credential request events. In such cases, advertisement of the events will occur as soon as the credential server has started up. The other cause for these events will be explicit action on the part of principals to effect delegation.

For a principal to be able to delegate a credential, they would need to acquire a particular privilege on their own EDSAC node. Using OASIS inferencing, the binding of parameters in this privilege indicates the target of the delegation, and the delegated credential itself. The existing form of the OASIS policy language suffices to support delegation – prerequisites of OASIS rules will simply become valid through external events rather than using local activation rules. To reduce unnecessary state in the publish/subscribe system it may be desirable to augment role definitions with an indication of whether rule evaluation engines should accept events leading to the delegation of certain credentials. We demonstrate the delegation of roles using our EDSAC prototype in section 7.4.3.

Sometime before a principal successfully acquires a delegation privilege, the EDSAC node involved must advertise credential forward events. This is necessary to initialise the publish/subscribe system event-routing state. It may be acceptable to advertise such events very soon before first publishing them. A more efficient strategy would probably be to advertise credential forward events as soon as principals successfully activate a role from which delegation privileges are potentially reachable. The most appropriate strategy will need to balance the latency of advertisements against the expense of the extra routing state if advertisements long preempt the events to which they refer.

Credential revocation

Credential revocation events occur when a role is deactivated, or a credential store discovers the invalidity of a previously forwarded credential. The publishers of these events will be, respectively, the sites at which roles are active, or credential storage services. Subscriptions to a revocation event pattern should occur whenever a given EDSAC node activates a role that has membership conditions within that credential pattern.

Again, publication of these events will eventuate due to both automatic and manual processes. Credentials will be automatically revoked when an EDSAC node learns these credentials have become invalid (e.g. through other revocations). The manual case involves principals activating a revoke privilege, similar to the delegate privilege discussed above.

Note that ending a user session is a special case of credential revocation. Because all of the roles activated within that session will need to be deactivated, it is more efficient to send a revocation message with a credential pattern that covers all session-based credentials (i.e. active roles), than to trace the transitive relationships between prerequisites, deactivating each in turn and then cascading.

Two-phase role activation

For the reasons discussed above, we treat the activation of roles differently from the 'discovery' of valid credentials. This is because role activation by some EDSAC node may trigger dynamic constraint violations. We instead divide role activation into two phases. A proposal phase, marked with a role activation request event, and a commit phase, marked with a role activation confirmation event. Interested parties may send role activation vote events between the request and commit phases.

The role activation request event contains an attribute indicating when this activation is intended to occur. This informs other interested EDSAC nodes by when they must register their agreement or disagreement. The primary cause of disagreement will be if this role activation would violate a dynamic integrity constraint. We can thus implement a distributed scheme to support mutually exclusive role activation [Kuh97, SZ97], or Sandhu's Transaction Control Expressions [San88]. Whilst it would be possible to include aspects of such distributed integrity checks using environmental predicates, this approach would be dangerous. It would risk race conditions occurring between nodes due to an impedance mismatch between the EDSAC network state and the state being communicated by the environmental predicates. It may be that an EDSAC node requests an extension of the deadline if determining the activation validity is a highly complex task (or involves human interaction).

Replies to the role activation request event come in the form of role activation vote events. The vote weight attribute is a numerical value indicating support (positive values), denial (negative values), or an indication that an answer is not yet available (zero). Any EDSAC node which does not care about the activation request at hand simply does not reply to it. (Although we have explicitly specified within our architecture a mechanism for weight-oriented conflict resolution (similar to XACML combining logic – see §2.2.4), our implementation does not make use of non-negative voting weights. In our tests, a role activation does not proceed if any other EDSAC nodes publish a negative role activation vote event.

After voting conflict resolution has been applied, if the role activation is deemed acceptable, a role activation confirmation event will be published, and the two-phase role activation is complete. The role activation confirmation event is required to allow further events to be triggered elsewhere in the EDSAC network – for example, if a more complex vote combination algorithm was used than in our prototype, an EDSAC node might register a negative vote, and if overruled, need to deactivate its own now-conflicting role.

In terms of advertisements and subscriptions, any EDSAC node at which a role needs to be activated that has dynamic constraints attached will need to advertise role activation request and confirmation events, and subscribe to role activation vote events. After this role is active, it will probably also need to advertise role activation vote events in order to participate in distributed conflict detection. In our prototype, we augment OASIS XML policy syntax to indicate whether dynamic constraint checks need to occur on a given role. When they do not, we revert to the much simpler protocol of publishing a role activation confirmation event as soon as a role activation that meets all the OASIS rule prerequisites is requested by a principal.

Our architecture can easily facilitate situations in which dynamic constraints are not monitored by all EDSAC nodes. For example, in a workflow management application it may be desirable to keep policy enforcement points lightweight, instead focusing constraint checking into a shared workflow monitor. In such cases the policy enforcement points would not need to advertise role activation voting events, since the workflow monitor node would do so on their behalf.

Action request and privilege vote events

The final pair of events required manage actions. Action request events cause remote EDSAC nodes to evaluate some item of policy. As indicated in table 6.1, action request events indicate a rule pattern, and may include a requester and target session filter. An action request event indicates that the given rule should be evaluated within a particular EDSAC session.

There are two main types of target for policy rules. They will either lead to a privilege (as in OASIS authorisation rules), or to some target role (as in OASIS role activation rules). If the rule leads to a privilege, the target of the action request event will evaluate whether or not the privilege is deemed to be true based on its local policy state, and will cast a privilege vote with an appropriate weight (our implementation only used positive and negative weights, for binary decision making). Of course if there is only one target evaluating a rule, this privilege vote event translates to a grant/deny reply to the action request. For the requested privilege to be granted, the publisher of the action request will need to have arranged for appropriate prerequisite credential forwarding messages to have been sent in advance to the subscriber. A privilege vote event will be published by the target node on its completion of rule evaluation. Depending on the nature of the privilege, it may be linked to an action at the target node (so the vote is an indicator of whether the action was initiated) and/or cause the requester to perform appropriate actions when it receives the privilege vote event.

In the cases where a rule leads to role activation, not all prerequisites need to have already been forwarded to the subscriber since, with a suitable inference engine, it can issue its own credential request events.

6.2.3 State spaces

The events presented in the previous sections build an architecture for both passing state and checking constraints within a distributed privilege management network.

We have avoided specifying how each EDSAC node will reach its policy decisions, and have instead focused on the mechanisms through which it passes its decisions to other

EDSAC nodes. Indeed, we envisage support of networks in which sets of EDSAC nodes have inference engines with very different capabilities. Note that in our tests we used a Prolog inference mechanism whose clauses are generated via XSLT transformations from OASIS-like XML policy files.

The true policy state is contained in the sum total of all the EDSAC nodes within an access control network. The events presented above allow each EDSAC node to maintain awareness of a much smaller subset of this state space. This avoids EDSAC nodes requiring undue quantities of local storage, and best models the lack of knowledge any EDSAC node has about what state will be relevant to it based on unknown future actions of its principals.

We have tried to ensure that, if conflicts do occur in the network, interested parties will be aware the conflicts have occurred, and if necessary initiate recovery actions subsequently.

6.2.4 Push or pull credential discovery

Push-based credential status updates (particularly revocations) are beneficial in that nodes depending on particular credentials do not need to spend time polling for changes in credential validity – an event will inform them when credentials are revoked.

However when evaluating policy rules, it is useful to have both push and pull options for credential discovery (such as in PERMIS [CO02a]). Thus we support both forms of credential discovery in EDSAC. Push-style credential discovery involves other nodes publishing credential forwarding events, and thus setting up local credential awareness, in advance of those nodes making role activation or privilege requests. This style of credential communication suits invoking privileges on lightweight Policy Enforcement Points (PEPs) – these enforcement points may not have sufficient computational resources to evaluate complex policy rules. OASIS membership conditions require implementations to support a push-style of credential update.

In pull-style credential discovery, when a policy inference engine is unsure of the state of a prerequisite credential it publishes a credential request event to discover it. Because pull-style discovery of credentials occurs using content-based event delivery, it allows for much better separation of policy concerns between different parts of an EDSAC network.

Different applications will also lead more naturally into one or the other form of credential discovery. EDSAC deployments in which users are highly skilled and work on particular tasks (e.g. clerks trained in a particular set of procedures) will probably know which privileges they need to request, and what roles should to be active for them to make these requests. Presented with a push-style interface, these highly trained users will be able to directly specify what requests they want the EDSAC network to check.

Pull-style credential discovery much better suits users who are ‘browsing’ a system and have no interest in knowing how the underlying policy works. In such cases, the discovery of a path through prerequisite credentials to a target will probably be left to the access control software. Note that by making preemptive credential requests, pull-style credential discovery may lead to violation of the principle of least privilege. Further, it is essential that these preemptive credential requests do not themselves trigger dynamic constraints in the system.

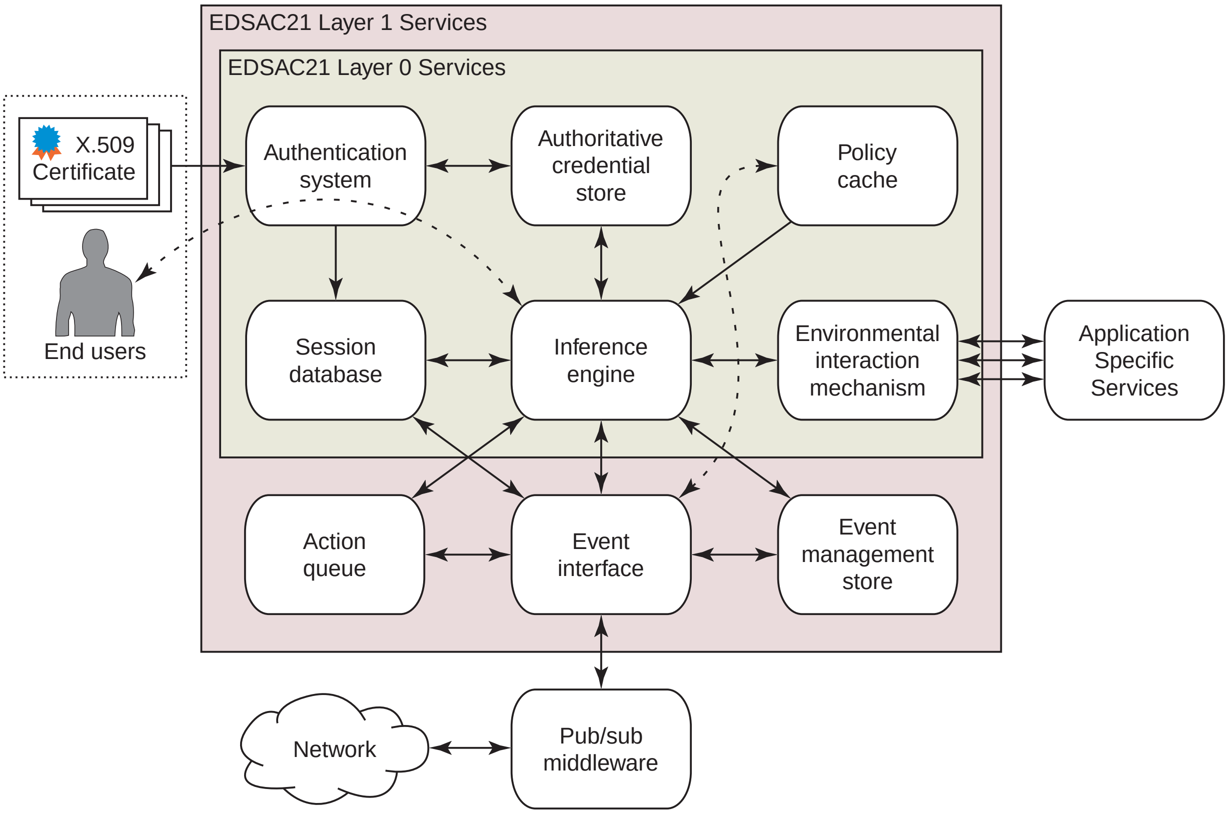


Figure 6.1: Components within each EDSAC node.

6.3 EDSAC nodes and networks

This section describes the nodes which send and receive the events described in the previous section, and the interconnected access control networks they form. An *EDSAC network* is a set of interconnected, but distributed, *EDSAC nodes*, each corresponding to a host for one or more protected services.

6.3.1 Components of an EDSAC node

An EDSAC node provides basic communication primitives to participate in access control negotiations. If we were to use the terms of the ISO/IEC 10181-3 standard, such nodes are most likely to participate in the network to play the part of a Policy Decision Point (PDP), a Policy Enforcement Point (PEP), or indeed perhaps both simultaneously. The components of an individual EDSAC node are as follows (see also figure 6.1):

Event interface. Every EDSAC node has an interface to an underlying messaging system. Many of the features discussed in section 6.4 require this messaging system to support topic- and attribute-based publish/subscribe expressions – in our testing we used the Hermes [PB02] publish/subscribe system.

The event interface is protected by the layer 0 logic at this node. In other words, policy rules need to be satisfied to gain the privileges required to publish and subscribe to events. Of course public access can be achieved through the use of null policy.

Event management store. Operating above EDSAC layer 0 requires that nodes lodge advertisements and subscriptions with the event delivery system. The event management store maintains these advertisements and subscriptions, and indicates whether they relate to this node in general, or to particular user sessions operating on it. Example subscriptions would include those relevant for monitoring context checks or other dynamic constraints, and the potential revocation or delegation of credentials.

Policy cache. When an EDSAC node is determining whether a request is authorised, the decision is based on rules held in the policy cache. Although the rule database at a node is stored persistently, we use the term *cache* to emphasise that the complete access control policy associated with a particular application is managed using EDSAC layer 2 logic, and is distributed across a number of such databases. Each policy cache contains the rules relevant to services running at the local node. Policy will evolve over the lifetime of a distributed application, and it is therefore important that this cache can be updated. Updates to the policy cache can be controlled by rules within it, in the manner of Administrative RBAC (ARBAC) [SBM99], or new rules can be deployed in response to high-level policy change by external action as necessary. Either path should allow policy evolution with minimal disruption to users.

Session database. As opposed to the comparatively static policy cache, the contents of the session database are expected to change frequently. The most basic requirement for the session database is to record active user sessions, and the role-activation state within these sessions. These two items cover the requirements of the OASIS model. In our EDSAC implementation we also store the state of the policy contexts used to check dynamic constraints.

In may be useful to go a step beyond OASIS and also store policy (i.e. sets of rules) asserted by trusted clients for the duration of some given session. This would facilitate dynamic service level agreements, since the policy rules to translate foreign credentials into local equivalents can be presented to an EDSAC node alongside the foreign credentials themselves. Whilst our EDSAC prototype supports this functionality, it has not been a focus of our evaluation.

Action queue. Two-phase role activation requires that certain events be deferred some period of time into the future. The action queue is where these potential actions are recorded. Negative role activation vote events may lead to the premature disposal of actions from this queue. Beyond role-activation, the action queue can also be used to manage roles which have limited-duration validity. When these roles are activated, appropriately deferred revocation events can be inserted into the action queue (e.g. to support obligations, as discussed in section 7.4.4).

Authoritative credential store. In addition to monitoring active role state, EDSAC nodes may have the capacity to store other credentials. The form of this credential store might well be an LDAP directory of X.509 certificates, although we used simpler local database implementations in our prototype.

Inference engine. All EDSAC nodes must contain some form of inference engine. Each layer of the EDSAC architecture will have different requirements however. At layer 0, the inference engine must make decisions entirely based on information it has locally (it behaves as a PEP). All privilege requests fall into this category. Since using an EDSAC node's event service API is a privileged operation, its access is also protected by such rules.

Layer 1 inference engine behaviour builds on layer 0, and thus if authorised can make use of the event service to determine a rule evaluation (thus involving other nodes in the EDSAC network). Layer 1 inference behaviour is consistent with that of PDPs.

Layer 2 of the EDSAC architecture does not have its own inference process, but involves design logic which will dictate the policy rules at lower layers. This will need to integrate application-specific knowledge. In a workflow application, for example, layer 2 logic may dictate that a principal needs to activate a specific role at a central workflow monitor before it will respond to any credential requests made during the evaluation of layer 1 rules.

In terms of OASIS, authorisation rules all sit at layer 0. The OASIS intuition is that authorisation rules only perform last-minute checks before an action is performed or denied. In EDSAC we guarantee such checks are last-minute by restricting their evaluation to locally-stored credentials. Role activation rules sit at layer 1, since it is reasonable that policy evaluation may need to communicate with other EDSAC nodes regarding the current prerequisite state for a given principal. Note that OASIS did not originally provide an indication of when remote sites should be contacted to verify credential state for role activations (i.e. pull based activation). A degree of inter-OASIS node pull-based credential discovery was added to CBCL OASIS, and uses the internal Simple Object Access Protocol (SOAP) interface between OASIS services. In EDSAC we support both push and pull style policy evaluation with our event set. However our prototype inference engine only uses credentials which have been pushed to it. This approach ensures we best maintain the principle of least privilege.

Environmental interaction mechanism. Environmental interaction is a combination of inference engine and policy cache functions. The local policy cache defines what external functions are available through the EDSAC call-out mechanism, whilst the inference engine is responsible for marshalling and un-marshalling the types and values of EDSAC rule parameters to and from the arguments of external functions. Environmental predicates may be used to introduce time considerations, external database lookup, or other application-specific criteria into EDSAC inferencing. They must be used with caution, since they reside outside the access control framework, and their semantics cannot be guaranteed. A way of managing the information flow when invoking such external functions is explored in [BEM03] – we classify the

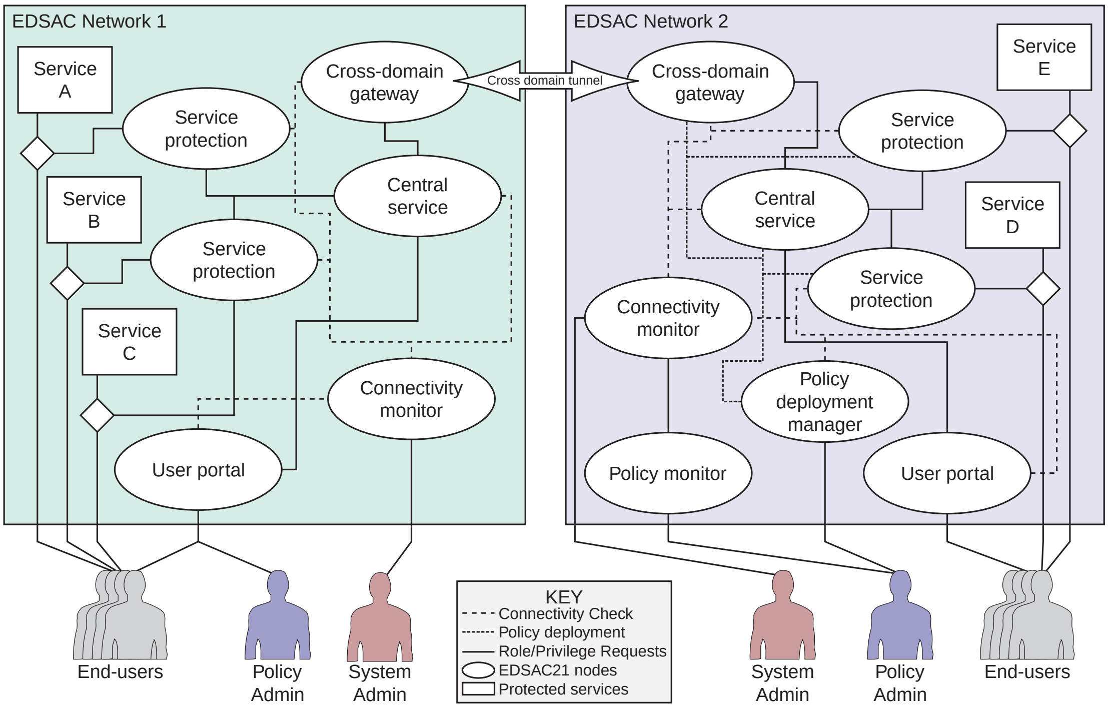


Figure 6.2: EDSAC node types in a hypothetical multi-domain application.

parameters of each environmental predicate so we can derive potential information paths.

Authentication system. Each EDSAC node requires a means of establishing the basis of trust on which to accept the authentication information or other data presented to it. Our prototype implementations have used X.509 certificates for secure authentication. Apart from providing an initial OASIS-style appointment from which to begin a session on behalf of a user, agreement on a common entry in the client and server certificate chains is expected to help in cases where a principal is not known to a particular EDSAC node; the intention is that they can present a signed translation policy to activate within their session, thus acquiring locally-defined credentials approximately equivalent to their foreign ones. We are also currently examining whether we can use trust evaluation systems as a basis for authentication.

6.3.2 EDSAC deployment structures

We have given an overview of the functional components at each EDSAC node, so next describe the structure of an EDSAC network (layer 1). The EDSAC nodes on a network are likely to perform different primary functions, even though they present a homogeneous layer 0 interface to privilege requests. Some example functions are described below (and are shown in figure 6.2):

User portal nodes. These are the initial point of contact between user software and the

EDSAC network. It is here that user sessions are created, and role and privilege requests made of other EDSAC nodes. Note that this process may well occur implicitly, covered by actions performed on behalf of the user by an EDSAC-aware application or web-interface, for example.

Central service nodes. These nodes handle a high number of requests, and have a rich environmental interaction capability, dependent on the need of the application in question. They operate primarily as PDPs in the terminology of ISO/IEC 10181-3.

Service protection nodes. Service protection involves minimal role-state storage, and limited inferencing. These EDSAC nodes are the last point in the EDSAC network; applications are likely to make privilege requests of such nodes before performing EDSAC-protected actions. They most directly relate to ISO/IEC 10181-3 PEPs.

Cross-domain gateways. In large-scale distributed services, it is likely that multiple EDSAC networks will cooperate. For audit and logistical reasons, a number of specific gateway nodes may be deployed to act as conduits between the two networks, performing credential translation as appropriate.

Connectivity monitoring services. Whilst we expect that underlying event-based messaging software will generally perform connectivity testing, there is a level at which the response to network events should be policy-based (this is explored in [BE03]). In early testing, we implemented connectivity monitoring at the EDSAC level, and thus could specify events to be sent when nodes became disconnected from the network.

Policy deployment managers. These are nodes whose primary job is to assert policy into (and retract policy from) other EDSAC nodes. Any significant-scale system will need such a service in order to manage policy on remote nodes.

Policy monitoring nodes. The underlying publish/subscribe system can allow certain authorised nodes to subscribe to changes in policy and access control state associated with current sessions. Such nodes enable security administrators to query and analyse the state of an EDSAC network interactively. This functionality may require event-based message types that are not specifically related to role-activation or privilege requests; these messages can also be used for logging the access control history of the system.

6.3.3 Losing connectivity to an EDSAC node

When connectivity to an EDSAC node is lost it is important that the sessions it hosts be considered invalid. Generally this will also involve revoking all roles it had activated within the EDSAC network. We discussed a scheme for tagging roles with an extended validity when connectivity is lost in [BE03] – our aim is to minimise the risk of undesirable mass role revocation in the face of any slight network glitch.

Detecting disconnection requires neighbouring nodes to notice the lack of heartbeat. Ideally this connectivity testing will be integrated into the event-based message delivery infrastructure. In our tests, we deployed an independent connectivity monitoring service.

Brokers on revocation event subscription paths must issue session revocation events if they assess that no alternative route can reconnect their subscription to the rest of the publish/subscribe network. Some policy operations can continue to operate even if an EDSAC node is disconnected (e.g. EDSAC layer 0 inferencing). Equivalently, the OASIS-like language used in our EDSAC examples does not need to initialise explicit revocation subscriptions on role activation rule prerequisites which are not membership conditions.

Were the entire EDSAC network to become partitioned, it is likely that layer 2 logic would no longer function correctly. Some EDSAC designs may lead to each partition being able to operate independently, but even in this case some functions will be missing.

6.3.4 Recovering EDSAC network synchronisation

A node that becomes disconnected from the EDSAC network will be aware of this situation due to a lack of received heartbeat events. It is likely that it will need to revoke numerous locally-active roles.

When a given node successfully rejoins the EDSAC network, it may have missed any number of EDSAC events. It will thus need to re-request activation of all of its roles. Moreover, it will need to issue specific credential request events to see whether it missed opportunities to vote against role activation requests.

It is possible that the event-delivery mechanism could queue messages that might have been of interest to a disconnected node. Such ideas are important for the support of mobility within publish/subscribe networks. In terms of security infrastructures, we currently assume nodes will not be mobile. Even so, a message queue would allow a reconnecting node to narrow its attention to the specific roles which have been activated in its absence rather than having to test all possibilities. We discuss the potential to do policy evaluation on mobile equipment in section 7.3, which relates to location-aware access control. Given mobility is not yet a wide-spread, stable feature of publish/subscribe systems, its inclusion in EDSAC would be premature.

6.4 EDSAC event-based security features

Layer 2 EDSAC logic builds applications from the policy services offered by EDSAC networks. This section explores the capabilities provided through the event-based approach to active security embodied in the EDSAC system.

6.4.1 Fast revocation

The main reason that the OASIS architecture was designed to sit on top of an event-based middleware was to support *fast revocation* of prerequisites tagged as membership conditions. Note that the CBCL OASIS implementation does not explicitly support fast revocation, although an equivalent effect can be included where necessary using environmental predicates.

Distributed access control architectures without integrated event-based infrastructure generally require pull-style checking of credentials. Either credentials are assumed to have validity for some duration after they have been verified (which is inherently dangerous), or credentials need to be re-validated immediately prior to their use as prerequisites (which is inherently inefficient).

Access control systems that use X.509 certificates should also use the X.509 certificate revocation list infrastructure to re-validate presented credentials. Whilst this allows for the revocation of previously issued certificates, it is not an ideal situation to digitally sign the truth of some statement, and then need to digitally sign the revocation of the previous truth. We argue that it is safer and more efficient to keep revocation tightly coupled with the access control architecture. In terms of safety, event-based management of revocation within the access control system allows actions to be taken immediately on credential revocation. For applications such as workflow management, where the credentials of principals may have short validities, it is inefficient to re-generate X.509 certificates given that the revocation certificates will often be more important that the original credentials. Of course, if credentials are to be presented outside the EDSAC network, they will need to be stored in a digitally-signed form.

6.4.2 Dynamic constraints

It is likely that distributed RBAC policy can only be expressed satisfactorily for very simple applications using static principal-role and role-privilege assignment (as would be implied by using the NIST $RBAC_0$ standard).

More powerful access control systems allow decisions to be based on the current state of the access control system itself. A common example of the type of policy which can be implemented on such systems is a dynamic separation of duties constraint (DSoD). For example, say a certain principal is able to acquire the role of a *poster-to* and a *moderator-of* an on-line message board. Systems supporting DSoD can ensure that this principal does not acquire both posting and moderating privileges simultaneously (which might permit unacceptable self-moderation). Simpler policy systems will need to revert to static separation of duties constraints, which involves partitioning the policy rules themselves so that no-one can self-moderate, with the undesirably restrictive side-effect that each principal can only ever acquire one of the two privileges.

The EDSAC event types presented in section 6.2.2 facilitate a range of dynamic constraint checks. Our use of a two-phase role activation protocol allows many nodes to participate in a layer 1 policy decision, without them requiring a complete copy of the current role-activation state (doing so would be inherently unscalable). Careful policy design is required to ensure that the dynamic constraints within an EDSAC network are efficiently managed however.

In order to experiment with dynamic constraints, we needed to extend the OASIS policy language. We integrated the specification of *policy contexts* (§4.4.2) into role-activation rules. These contexts are used to define named exclusion groups in which only a certain number of roles are permitted to be active at any one time.

This mechanism allows for cardinality constraints on role activation. Dynamic separation of duties can be modelled by setting the cardinality constraint to 1 for some collection of roles. More complex dynamic constraints can be implemented by using more detailed inferencing logic at the EDSAC nodes participating in conflict detection. We believe our two-phase role activation protocol will be sufficient for most potential dynamic constraint requirements.

6.4.3 Workflow

The ability to manage dynamic constraints facilitates integration of many aspects of workflow management into the EDSAC system. Workflow management systems (see [Sch99] for an introduction) specify how a particular task may be accomplished through performing a series of sub-tasks. Unlike the usual access control perspective, in workflow management it is usually permitted that there be a number of instances of a particular task in various stages of completion, involving various sets of principals. For each such instance, the workflow specification defines the various restrictions on task order and the principals authorised to carry out each of the sub-tasks. We explore workflow management in detail in section 7.4 and use it to evaluate our EDSAC prototype.

6.4.4 Conflicts

There has been a large amount of research into conflict checking within RBAC systems (e.g. [LS99, MS93, SM02]). In the EDSAC system we have described, conflicts may arise in both layer 1 and layer 2 logic. The latter involves application-specific behaviour, and is not discussed here (e.g. an application with multiple credential store sites should ensure they remain synchronised). Conflicts in layer 1 may occur through the simultaneous lodging of requests.

Because we maintain course-level overall EDSAC synchronisation using a network tick, many problems with delayed and out-of-order messages are simple to manage. The network ticks provide a means to judge the 'age' of an event, thus a cut-off can be set relative to the current tick prior to which old events are dropped. The ticks also provide an ordering for events should events arrive near to each other that, for example, activate and deactivate the same credential.

We need to pay particular attention to situations in which a conflict occurs within the same network tick. Of the events presented in table 6.1, the only events that pose a problem in this regard are the role activation events, since they can trigger constraints. The role activation vote and confirm events do not pose any problems, since the session identifiers they contain make their targets clear. Role activation request events require a conflict resolution protocol however.

Let us examine the situation in which two independent EDSAC nodes a and b initiate a potentially conflicting role activation request event. Say a issues its request at time A_r publishing its intention to activate a role at time A_a, and that b does the equivalent for times B_r and B_a. We consider a to have a higher priority in terms of our resolution

```
addEventType(typeowner,eventType)
removeEventType(typeowner,eventType)
subscribeType(subscriber,eventType,callback)
unsubscribeType(subscriber,eventType)
subscribeTypeAttr(subscriber,eventType,filter,callback)
unsubscribeTypeAttr(subscriber,creds,eventType,filter)
advertise(publisher,creds,eventType)
unadvertise(publisher,creds,eventType)
publish(publisher,creds,event)
```

Figure 6.3: Hermes API presented to EDSAC applications

protocol either because A_a is earlier than B_a, or if $A_a = B_b$, then the numerical identifier of a's session is lower than that of b.

The conflict situation will arise when B_r is not later than A_a but is later than A_r. This is because a may well issue a negative vote against b's request, but cannot do so while its own role activation has not yet been finalised. It is possible this conflict could be avoided if B_a is sufficiently later than A_a to allow a to raise its conflict vote, but this would also require a to cache such potentially conflicting situations until time A_a.

We take the simpler approach of ensuring in this case that b waits to re-lodge its event at or after time A_a. We require that b puts itself in the position of knowing about the conflict situation, by its subscription to role activation vote events being sufficiently general to receive events targeted at a as well as those targeted at itself.

6.5 EDSAC application services

Given that all EDSAC protected networks will include a comprehensive publish/subscribe event delivery system for security events, there is no reason not to offer publish/subscribe services to applications.

The Hermes broker API usable by EDSAC applications is listed in figure 6.3. Note that Hermes was implemented in Java, and hence these methods use Java types. In particular, this means that the `callback` provided in the `subscribe` and `subscribeTypeAttr` methods must be references to a class to which Hermes can deliver an event through method invocation.

In our tests, we interfaced Hermes to Prolog using local TCP/IP connections. The Hermes classes modified to present a local TCP/IP API can thus identify local publishers and subscribers through the port numbers they are connecting to, and use a simple text-based method-selection and attribute-passing protocol.

To provide these communication methods to applications, we pair each one with an EDSAC privilege request. These privileges are 'triggers' in that if they are granted, the

EDSAC system will also perform the requested publish/subscribe method call on behalf of the application.

A similar interface to that in figure 6.3 was presented in [BEP$^+$03]. Unlike that model, however, we do not need to explicitly pass the credentials used to request method invocation, nor do we need to provide methods which connect and disconnect nodes to their local broker. The broker connection methods are unnecessary since an application's interface to an EDSAC node is also an interface to a local event broker. When an application makes a privilege request for one of the Hermes API calls, the user session they are in will determine their active credentials, and thus the success of their request.

6.6 Federating EDSAC networks

Federations of EDSAC services can occur in two main ways. If administrative domains are willing to use consistent policy semantics, and to share the publish/subscribe broker network, they may be able to achieve federation without public disclosure of all information, using the attribute encryption we present in section 6.7.

If explicit conversion is required between policy and/or event semantics, or the federation requires a controlled set of gateway nodes connecting the different EDSAC networks, we need to form the federation using layer 3 logic.

Under these circumstances, querying credentials and checking for role activation conflicts necessarily takes longer. Let us assume we are federating EDSAC networks a and b. We recommend that in a hierarchical federation, the level above this publish/subscribe layer also uses the publish/subscribe paradigm for event dissemination – but with a considerably longer network synchronisation period.

Let us say the period of a's network tick is τ_a and the period of b's network tick is τ_b. These individual EDSAC network periods should be set based on two factors. First is the maximum acceptable computational delay in determining the solution of a policy rule. The second factor is the maximum expected time it takes for an event to reach all nodes in an EDSAC network. The period should be set to the maximum of these two quantities.

For example, say a given node n decides to attempt role activation at time T_n, and sends their role activation request event at that time. Due to there being some skew in the distribution of heartbeat events, we can only safely assume all nodes have received this event at the end of time T_{n+1}. Thus all nodes have the opportunity to consider the request for the time unit T_{n+2}. Conflict votes will be sent back to n during time unit T_{n+3}. Thus the role can be declared active starting at time T_{n+4}. Checking for role activation constraints across a federation will require significantly more time. It is for this reason that the role activating node can choose longer network time periods during which their role activation can actually take place.

We suggest setting the parent publish/subscribe layer's synchronisation period to the maximum of τ_a and τ_b, which we will call τ_f. We return to our example of a activating a role at time T_n, but now consider this conflict involving the whole federation, and thus the length of each time unit T_n being τ_f.

At the end of time unit T_{n+2} we assume the gateway from a to b will have calculated whether it is necessary to forward the request. By the end of time T_{n+3} we assume the event will have reached all interested parties in b. Conflict votes in b will be computed during time unit T_{n+4}. These will reach the gateway by the end of time T_{n+5}, and all interested nodes in a by the end of T_{n+6}. Thus a can safely declare the role active at time T_{n+7}. It should convert this time, rounding up to the next time unit in local period terms. This approach can be repeated at each parent layer of a hierarchical federation. Of course the more that is known about node behaviour within a federation's networks, the more that the federation network tick period may be safely reduced.

6.7 Secure publish / subscribe systems

This thesis focuses on building active security using a publish/subscribe system. We explained above how EDSAC can be used to provide access control to a publish/subscribe API. One key concern is the security of the publish/subscribe system itself.

Generally publish/subscribe research has focused on making event dissemination easier, rather than keeping tight control over it (notable exceptions being [Mik02, WCEW02] and our own research [BEP+03]).

Before discussing our approach for securing event communication, we emphasise that it is important to be clear about the threats that we are trying to protect the event delivery infrastructure against. In our prototype we assumed that the EDSAC nodes themselves could always be trusted. We have not chosen to focus on protection against hijacked EDSAC nodes since we felt that the requirements for doing so would only become clear when EDSAC is used in a large-scale real-world deployment.

One way in which some extra event-level security can be added is by 'signing' the most important events. If these events represent privileged actions, the targets might only carry out the actions if the message signatures can be verified. Using a PKI approach will require that the target nodes are able to reliably discover the public key of the sender. The technology to sign events is readily available (e.g. through SSL/TLS and XML signatures). Such an approach would not provide protection for the contents of messages, however.

Restricting who is allowed to publish or subscribe to events in a publish/subscribe system first requires some degree of link-level security. In Hermes, TLS can be employed as a wrapper for the raw event exchanges between brokers. Secure access can be maintained provided there is a reliable process for determining whether or not a node should continue to trust its immediate neighbours. Clients use the publish/subscribe system by connecting to local brokers. In Hermes a publisher or subscriber makes an explicit connection to a local broker. In the EDSAC framework discussed in section 6.5, local brokers are those from which EDSAC principals will request event delivery API privileges.

The publish/subscribe system security discussed so far can be viewed as a boundary of access control to the message delivery framework. This situation is illustrated in figure 6.4.

In collaboration with other Opera Research Group members, we have recently devel-

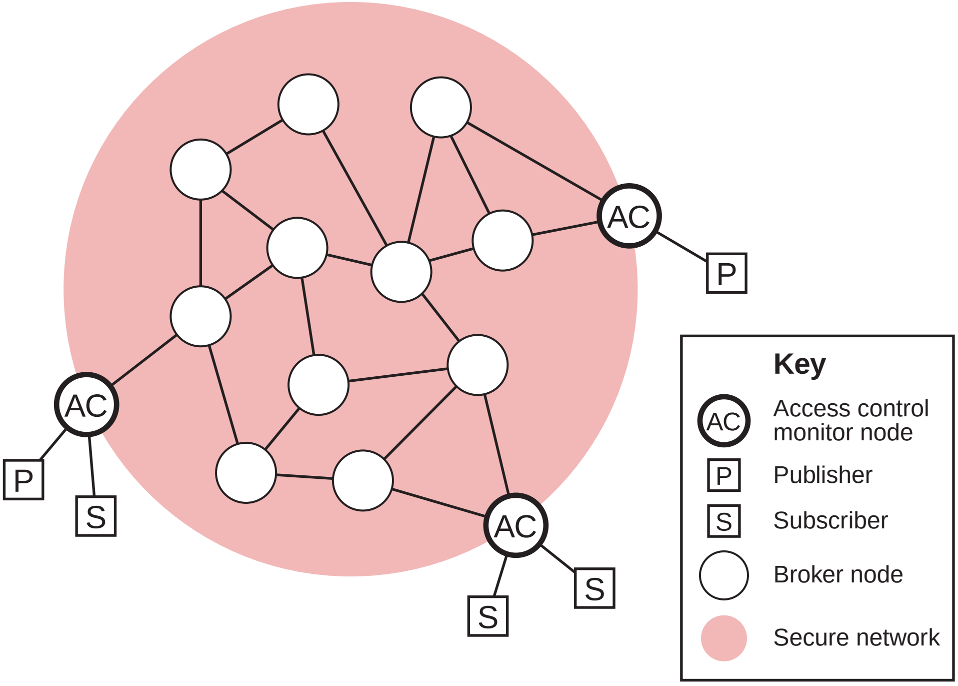


Figure 6.4: Boundary access control to publish/subscribe systems

oped an approach more powerful than boundary access control, in which we can selectively protect subsets of events using attribute encryption. We can then support events travelling through a federation of event brokers from different administrative domains. Such a network is shown in figure 6.5.

Attribute encryption requires an extension of event type definitions. Each event type needs to indicate the set of named symmetric encryption keys permissible to use for encrypting its values (including a null key which will cause the attribute to be included unencrypted).

During event publication, each attribute will be coupled with the set of encryption keys that should be used to protect its value. The publish/subscribe system will encode copies of the attribute so that any subscriber with any of the keys indicated in the set at event publication will be able to decrypt that attribute's value.

Note that we never disclose the encryption keys to end users. They refer to the keys by name, and their local broker retrieves the key values from one of the key moderators of their administrative domain.

The concept of administrative domains leads to two main requirements in an attribute-encryption supporting publish/subscribe system. Firstly, the policy rules at the edge of the event delivery system need to be consistent – in other words principals with similar credentials should be able to acquire comparable privileges at any EDSAC node within an administrative domain. The other main interpretation of an administrative domain is that nodes within it are permitted access to attribute encryption keys which belong to that domain.

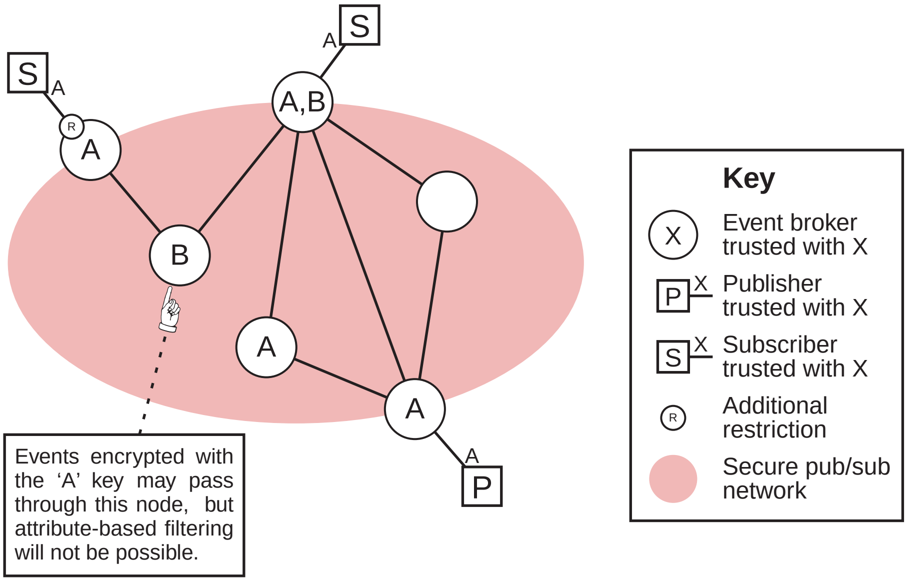

S
A
R
A
A,B
A
S
B
A
A
A
P
Events encrypted with the ‘A’ key may pass through this node, but attribute-based filtering will not be possible.
Key
X
Event broker trusted with X
P
X
Publisher trusted with X
S
X
Subscriber trusted with X
R
Additional restriction
Secure pub/sub network

Figure 6.5: A multi-administrative-domain publish/subscribe system using attribute encryption

When a local broker is required to use an encryption key they have not already cached, this broker will use the underlying message system to find the event type definition. These definitions will contain references to locations of the encryption key owners for this domain. The local broker then tries to contact one of these encryption key owners. The encryption key owner will check that the public key for this local broker is in the list of brokers within this administrative domain, and if so deliver the encryption key over a secure connection back to that local broker. Alternatively, the encryption key may be asymmetrically encrypted using the local broker's host key (which is used for establishing TLS connections), which can then reach the local broker through alternative channels (such as being delivered by the client).

Brokers route events based on their type and attributes. Thus if a given broker cannot decrypt an attribute, it cannot perform any filtering based on the specific subscriptions to it. The subscriptions to events with encrypted attributes appear as subscriptions with no restrictions on such attributes. This will lead to some redundant routes and state in the publish/subscribe network, but the type-based restrictions still operate at full effectiveness, as do subscriptions to publicly-accessible attributes.

The net benefit is that independent administrative domains can agree to share their event delivery infrastructure in order to decrease delivery latency and increase resilience to failure.

6.8 Trust-based access control

Before we present our EDSAC implementation (chapter 7), we discuss emerging techniques for the authorisation of privileges that do not rely on explicit prior knowledge of the principals involved. In EDSAC, trust-based access control will have an impact on both the inference engine and the authentication system.

Capability systems already provide a means to authorise principals without requiring identity-based authentication. Trust-based access control differs from these systems in that the capabilities presented by a principal do not need to relate directly to privileges. Instead a decision is based upon negotiation and the weighing up against the risk of the access control system making an incorrect decision in a particular case.

The primary motivation for this type of trust-based authorisation is to manage situations in which it is impractical to have shared credential information. Thus entities 'roaming' from their normal home are able to negotiate privileges without the need to form connections between the credential databases used by each environment. Of course not knowing these principals well may significantly downgrade the level of privilege they will receive in a 'foreign' environment.

One application domain in which such schemes will be very useful is in ubiquitous computing environments. Ultra-thin clients such as mobile phones and PDAs are not likely to be able to support the same policy evaluation mechanisms employed by wired computers, possibly because of a lack of access to any large-scale centralised security infrastructure. It would be inappropriate to require that such communities rely on manual configuration.

We consider three levels at which trust-based decisions can be incorporated into access control systems. At the most basic level, a multi-stage process of credential disclosure can be encoded into policy rules. In RBAC this could be modelled by having a number of rounds of role activations between two negotiating parties. At the success of each round, the next round permits more sensitive role activations and privilege requests. Such credential disclosure is discussed in section 6.8.1.

The next level up in complexity builds on deterministic credential negotiation by incorporating statistical uncertainty into the process. We present one such approach to statistical credential disclosure in [EM03].

The most complex form of trust-based access control involves integration of a trust and risk engine into the access control inferencing process. Each decision weighs up the amount of evidence to support the principal's right to perform a given action, with the risk that they will abuse the trust which would need to be given to them. We have recently examined how this type of trust-based access control can be effected by combining OASIS with the trust and risk engines developed as part of the SECURE project [DBE+04].

6.8.1 Trust negotiation protocols

There has been significant prior research into schemes which allow gradual exchange of credentials. A good overall review of trust negotiation research is provided in [Yao02]. Yao describes a number of existing key-centric trust management systems, in particular tracing

PolicyMaker's [BFL96] evolution into KeyNote [BFK99] and the Internet Engineering Task Force's (IETF) Simple Public Key Infrastructure (SPKI) [EFL+99]. Most of these schemes rely on asymmetric digital cryptography to allow a party to prove they own the private key of a key pair in response to a challenge issued by another party. Determining whether a party has a private key is trivial if the challenging party has a copy of the corresponding public key – they can simply send a challenge message that is encrypted using this public key. Thorough methodologies for incremental trust negotiation with limited disclosure are introduced in [BB03, WL02, YWS01]. These present mechanisms by which the policy dictating negotiation is itself gradually revealed as increasingly sensitive credentials are shared and verified by the two parties. Bertino, in [BFS03], presents a similar scheme for negotiation in the XML domain.

Many of the research mentioned above goes further than simply describing how the sequences of credential exchange for trust negotiation are structured. Important considerations covered by Winsborough in [WL02] for example, include providing a bound on how many steps of a protocol are required to reach a conclusion, and indeed, showing that all possible negotiation paths reach the same valid conclusion.

6.9 Conclusion

This chapter has presented our Event-driven Distributed Scalable Authorisation Control model for the 21st century (EDSAC21 or just EDSAC). We have discussed how the four layers of the EDSAC architecture build upon each other to facilitate an active, scalable, distributed access control infrastructure. We also discussed potential trust-based alternatives to identity-based authentication.

In terms of the principles we set out in chapter 3, we have significantly evolved the OASIS model with respect to managing dynamic constraints. Because credentials do not need to be stored as digitally-signed credentials we are able to dynamically change the duration of their validity.

In our prototype implementation, our policy language uses an extended version of the OASIS language and in doing so preserves its bounded policy evaluation and environmental interaction properties. Also, the EDSAC layer 0 components remain separate from the services they protect.

We allow the grouping of principals by referring to them using pattern-matching attributes rather than explicit identifiers in EDSAC events. The next chapter examines our OASIS policy language extensions in more detail, and discusses the remaining dynamic privilege management principles.

We explained how, in addition to providing dynamic access control to services at a site, we also provide a secure publish/subscribe event delivery interface (since this event messaging infrastructure is needed to support EDSAC operations anyway).

Having discussed the EDSAC architecture, we are now in a position to demonstrate in the next chapter the policy features it facilitates.

7 EDSAC case studies

This chapter examines our EDSAC implementation and the test cases we developed to demonstrate particular features of EDSAC architecture.

We begin by introducing our SWI-Prolog development environment in section 7.1, and the mechanism we developed to support network-based inter-process communication. Using these Prolog servers as our basic building block, section 7.2 describes our implementation of each of the EDSAC design layers and the EDSAC event system. We also discuss a Java-based authentication manager that reuses the CBCL OASIS X.509 certificate management code – we thus allow users to present X.509 certificates to determine initial OASIS-style appointments in EDSAC.

We present our research into location-aware access control in section 7.3, and test our implementation in a basic experiment in which location data is protected using distributed access control.

Section 7.4 presents our work combining access control and workflow systems. This case study demonstrates EDSAC enforcing dynamic constraints in a distributed environment.

Both experiments demonstrate a need for dynamic, collaborative, context-aware access control decisions. EDSAC meets this need by building on the foundation of OASIS, but placing particular emphasis on active access control. The integration of event-based communication is required to support the necessary credential delegation, revocation and dynamic constraint checking in both experiments.

7.1 Prolog as an EDSAC implementation language

This section describes the Prolog environment in which we created prototypes of the various layers of the EDSAC architecture.

Our choice of Prolog as an implementation language was based on its conciseness, its associative searching, and the simplicity of its type system. The associative aspect of Prolog provides a very interesting parallel to content-based message delivery. In the same way that the subscribers of an event are discovered by using publish/subscribe coverage relationships, searches on the Prolog internal clause database use term unification to find matches. For efficiency most Prolog engines will provide an extra degree of indexing to speed clause searches, for example, using a hash on the first argument of a functional term (as in SWI Prolog).

Prolog is an International Standards Organisation (ISO) standardised language [ISO95]. Its fundamental language definition is beautifully simple and yet extraordinarily expressive. This is in stark contrast to the world of Enterprise JavaBeans, where the standards involved in the Java language itself, let alone configuration of the application servers involved, have grown beyond the working-set size of many human minds.

Given that we were spanning XML, Java, and ASN.1 types, for experimental purposes we avoided a large amount of marshalling code by dealing in the lowest common denominator: character strings. Where convenient, we use tagged functional terms in Prolog to approximate some aspects of variable typing – that way unification of incompatible terms will fail, although there is no specific notion of this being due to type mismatch.

Our EDSAC prototype does not use XML policy files directly (e.g. via a Prolog XML parsing library such as the SWI-Prolog SGML/XML parser [Wie01]). Instead we use the Apache XML Project's Xalan [ASFb] XSLT framework to generate Prolog predicates from the original policy files (directly applying Prolog-style inference and search on XML files was tempting, but our work in [ESW02] showed that this rapidly became unwieldy). For policy without dynamic constraints, OASIS XML policy rules map almost directly into Prolog clauses. These automatically-generated Prolog files import our EDSAC policy management library to define common predicates and manage network communication.

7.1.1 SWI Prolog

The actual Prolog engine used to deploy EDSAC is SWI Prolog [Wie87]. It is an Edinburgh/ISO Prolog compiler released under the LGPL licence, and is an ongoing project of the Department of Social Science Informatics at the University of Amsterdam. Some of our previous development work had used SICStus Prolog [Car95] and Ciao Prolog [CHV00, CH01], but these Prolog implementations did not offer a sufficient number of additional features to justify choosing them over an engine with fewer potential licensing difficulties.

The core SWI Prolog engine is a modified Warren Abstract Machine (WAM). This leads to greater efficiency compared with simpler stack machine implementations. For example, tail recursion will not be performed – with its associated linear stack usage – when such structures are, in effect, iterative. An excellent overview of WAM mechanics is provided in [AK91].

We are aware that there are other logic-based languages that provide far more efficient execution. This is usually facilitated through the language requiring meta-information about predicates; for example, the number of solutions that predicates will return given certain classes of inputs (e.g. Mercury [SHC96]). In our environment, however, we require there to be a run-time component able to parse and interpret predicates – in EDSAC this mechanism allows us to perform certain types of dynamic policy modification. It will eventually be sensible to replace our Prolog implementation with one focused on speed and memory efficiency. For the moment, we benefit from being able to work in an interpreted environment where we can examine the current predicate stack and modify each Prolog database with maximum flexibility.

The environmental interaction facilities of Prolog are provided using meta-predicates. These predicates are not evaluated within Prolog itself, but instead provide an interface to the rest of the operating system in which the host Prolog process is running. As an example, ISO standard meta-predicates allow interaction with file input/output.

Unfortunately, Prolog inter-process communication (IPC) is not a standard part of the language. Because we did not want to become dependent on libraries specific to particular Prolog implementations, we make use of only one non-standard (but widely implemented) interface; TCP/IP sockets.

By using sockets we can effect both local inter-process communication and remote procedure calls (RPC). Apart from connection initiation, sockets appear as conventional streams to our Prolog code. Indeed our prototype could also be implemented using file streams connecting to named Unix pipes; although this would have standardised the Prolog, it would have precluded deploying our EDSAC prototype easily on computers running the Microsoft Windows operating system.

This stream-based communication mechanism was tested to work with a local Hermes broker to provide the EDSAC layer 0 event interface (§7.2.1), and was used to accept connections from our Java-based authentication system (§7.2.5).

7.1.2 Prolog term-servers

Given the proliferation of high-speed global networking over the past decade, many protocols have developed to allow languages to perform operations on remote machines as if they were local. Languages such as ‘C’ and ‘C++’ do not include such concepts of services within their language definitions, consequently systems such as CORBA IDL [OMG02] provide a means for describing how to marshal data across network calls, and CORBA libraries facilitate actual network interaction. Local method ‘stubs’ then provide access to remote procedures. Newer languages such as Java include messaging concepts within their (vast) APIs, along with the serialisation notions required to transmit language data across networks.

Whilst it is not a complicated idea, nor a completely original principle, the engineering of EDSAC draws attention to how well Prolog can be fitted with the notion of remote procedure calls. This benefit comes from the ease of serialising Prolog terms into strings of characters, combined with Prolog’s own unification of the concepts of program and data into one syntactical form. Indeed we gain the power of not only performing remote procedure calls, but also remote procedure execution (i.e. the passing over the network of code to be executed in a remote context). Languages such as Java can serialise their own program code, and thus have it evaluated remotely by a willing site, but eventually this approach becomes complicated through naming problems – a remote Prolog engine, by contrast, provides access to any global data or predicate through its unified internal database. Throughout our engineering work, we found it extremely useful to be able to switch between sending protocol-oriented predicate calls, and defining remote functions that are needed at runtime.

Our basic term-server module requires the use of network sockets, and thus is not completely ISO standard; socket primitives are not a core meta-predicate, although they are present in all the major Prolog engines with which we have experience. Stream-oriented TCP/IP sockets are sufficiently standard across a wide spectrum of operating systems to mean that, at our level of use, the differences between Prolog socket libraries are only syntactic. We note that it would be possible to define a text-based protocol by adding source and target headers and footers (e.g. an 'end of stream' message) into text streams, so that a wrapper environment could perform communication on behalf of Prolog. Whilst this would allow the Prolog side to be fully ISO Prolog compliant, the benefit in doing so seems minimal.

7.1.3 Term-server communication

Before describing our prototype's EDSAC layer 0 interface, we examine the underlying *Prolog term-server* on which it is based. These term-servers facilitate the execution of predicates on a remote Prolog process.

As with most other network servers, our Prolog term-servers bind to a given local socket and listen for incoming connections. To simplify the logic within Prolog, it is assumed that handling any given request is computationally cheap. Thus multi-threading or process forking is not required since the single Prolog process returns sufficiently quickly to a listening state. We remove this restriction by adding a multi-threaded Java-based communication front-end (§7.2.5). This Java wrapper parses X.509 certificates (since this capacity is built into the Java API, and would be unusual to see in a Prolog library). Note that the SWI Prolog socket library (care of the operating system) does, itself, provide a limited amount of buffering when accepting connections.

When a connection is received, all the caller's requested executable terms are read in and parsed by the Prolog engine. Parse errors cause terms to be dropped without notice. Note that as term serialisation will usually be performed by ISO Prolog meta-predicates, such parse errors will not occur during normal interaction.

Since the network communication actually uses TCP and not UDP sockets, there is a duplex connection to the sender of a term-stream. We do not currently take advantage of the potential to have a dialogue between a term-server and a caller. Although it would be an easy modification to make, we are modelling asynchronous rather than synchronous RPC. However, it is possible for a reply to be sent to the client who made the original request. The higher layers of the EDSAC architecture generally perform one-way event-based communication, so the default behaviour is to close the client connection as soon as the terms have been read. This allows terms sent to a particular node to perform further outgoing communication from that node. We provide a special prefix term to indicate that the term-server should leave a bidirectional network connection open, although this feature is not needed in our EDSAC prototype.

Two modes of operation are provided for handling term-list failure. Generally, the EDSAC use of term-servers will attempt to evaluate all terms, even if they fail. Strictly speaking, the term list evaluation does ‘fail’, so that complete backtracking is performed. When this is not desirable, terms can instead be evaluated so that they fail without backtracking.

This Prolog term-server notion has been useful outside the EDSAC project. The rule evaluation engine within the Cambridge Policy Analysis and Checking Environment (CamPACE) now also uses a Prolog term-server to perform its computations. CamPACE is most recently documented in [AEB04a, AEB04c, AEB04b], but is strongly based on the Electronic-commerce Development and Execution Environment (EDEE) project before it – see [AB00, AB01c, AB01a, AB01b, AB01d, AB02b, AB02a, AK02, Abr02, AEB02c, AEB02a, AEB02b]. Our work on CamPACE and EDEE has been largely independent of our access control research.

7.2 Implementing EDSAC

For EDSAC use, the term-server concept described in section 7.1.2 needs to be augmented in two ways. First, we need to provide a secure interface to the underlying term-server that limits the set of potential operations any client can request of this EDSAC node. This is effected by the Java security interface discussed in section 7.2.5. Our Java interface also provides the EDSAC layer 0 authentication manager.

The second consideration is that requests can arrive at any EDSAC node either through local connections (operating at layer 0), or through the publish/subscribe system (layers 1 and above). For our testing, we demonstrated that we could integrate our Prolog systems with a variant of Hermes that uses a text-based TCP/IP protocol that is similar to our term-servers (see section 7.2.1). However, because support for Prolog functional terms is not native to Hermes, the testing we present in sections 7.3 and 7.4 uses a Prolog-based publish/subscribe system emulation.

The sections below indicate how we implemented each layer of the EDSAC architecture in our prototype. We first describe our predicate interface to any non-Prolog publish/subscribe system.

7.2.1 Publish/subscribe system interface

The EDSAC layer 0 interface does not depend on publish/subscribe event delivery for rule evaluation, but it does use the interface for publishing audit messages. Its predicates are listed in figure 7.1 as they would unify with Prolog queries. These predicates map directly to Hermes API calls via our Hermes TCP/IP interface. It is for this reason that a notion of attribute type is included, although we do not use Hermes type features in our Prolog prototype. We normally refer to EDSAC attributes in code using the prefix `Attr`, whereas we use `PSAttr` for publish/subscribe system attributes.

The `PSAttrType` list contains terms of the form `attr(Name,Type)`, where `Name` is an attribute name, and `Type` is the type of that attribute (and is currently always `String` for our prototype). The `PSAttrValue` lists consist of atoms corresponding to the attribute

```
psAddEventType(TypeName, [PSAttrType|List])
psAdvertise(TypeName)
psPublish(TypeName, [PSAttrValue|List])
psSubscribe(TypeName, ReplyPort, [PSAttrFilter|List])
```

Figure 7.1: Publish/subscribe interface predicates

```
credReq(SrcSession, DstSession, DstContext, CredPattern, Deadline)
credForward(SrcSession, DstSession, DstContext, CredPattern)
credRevoke(SrcSession, DstSession, DstContext, CredPattern)
roleActReq(SrcSession, DstSession, DstContext, CredPattern, ActionTime)
roleActVote(SrcSession, DstSession, DstContext, CredPattern, VoteWeight)
roleActConfirm(SrcSession, DstSession, DstContext, CredPattern)
actionReq(SrcSession, DstSession, DstContext, RuleID)
privVote(SrcSession, DstSession, DstContext, PrivID, VoteWeight)
```

Figure 7.2: Prolog equivalent terms for EDSAC events

types, listed in the order they are presented in the addEventType predicate. Finally, the PSAttrFilter terms are of the form filter(Name,Pred,Value), where Name is an attribute name, Pred is a publish/subscribe predicate to be applied to this name, and Value is what the attribute will be compared to. In our prototype we always use equality within Hermes, and Prolog unification within our Prolog publish/subscribe emulation.

EDSAC events

The events required to support EDSAC security functions are shown in figure 7.2 as Prolog terms. The semantics of these events were discussed in section 6.2.2, and relate specifically to the contents of table 6.1.

There are a number of potential sources for events in EDSAC. All events may be delivered to a node through the publish/subscribe interface. Otherwise events are likely to be caused by explicit user requests, by requests within the policy definition, or by actions of the policy inference engine itself. We have categorised where we expect that EDSAC events originate from in table 7.1.

To illustrate the distinction between each event type source in table 7.1, consider an EDSAC node operating equivalently to one of our CBCL OASIS NHS EHR portal sites. When the **user** connected to this node requests a particular record, they are requesting the portal software to issue an actionReq event targeted at an Index Server node. In the case of events sourced from **policy**, a given user's privilege may lead to the delegation of one of their credentials (§7.4.3). This delegation would be effected by issuing a credForward

Prolog term	user	policy	inference engine	event system
credReq	Yes	No	Yes	Yes
credForward	Yes	Yes	No	Yes
credRevoke	Yes	Yes	Yes	Yes
roleActReq	No	No	Yes	Yes
roleActVote	No	Yes	Yes	Yes
roleActConfirm	No	Yes	Yes	Yes
actionReq	Yes	Yes	No	Yes
privVote	No	Yes	Yes	Yes

Table 7.1: Expected EDSAC event sources

event. If an action request instructs a given EDSAC node to perform role activation, the **inference engine** itself will issue `roleActReq`, and `roleActConfirm` events. The nodes which manage dynamic constraints over these roles will receive these events via the **event system**.

The use of all of these events is demonstrated in section 7.4.3. Within Prolog we unify incoming and outgoing requests with the following two predicates:

```
edsacReceive(Event)
edsacSend(Event)
```

The `edsacReceive` predicate is the entry point for all EDSAC requests at this node. Valid `Event` bindings will be one of the forms listed in table 7.1, and will thus indicate the action this node should take. Similarly the `edsacSend` predicate is used to publish events either through our Prolog publish/subscribe emulator, or via an infrastructure such as Hermes, using the predicate interface described in figure 7.1.

Using the Hermes TCP/IP interface

Our integration with Hermes uses a text-based TCP/IP interface designed for the convenience and accessibility of its loose-coupling, rather than its efficiency (XML encodings are no less verbose!). The details of our protocol are provided through an example. Consider a node that wants to be notified on local port 7002 if any role activation request event is published relating to a particular type of doctor role. Say the doctor roles of interest are of policy version `v1`, with a first attribute equal to 12. In terms of Prolog unification, the doctor roles of interest will match `role(doctor,v1,_,_,[12,_,_],_)`. We thus can represent the event we are interested in using the Prolog term shown in figure 7.3.

Because Hermes does not currently contain 'native' support for matching on functional terms, we defined a system of attribute names to be used to map to and from such terms. Through 'name mangling', the `role` term within this credential request event becomes the restriction on Hermes attributes shown in figure 7.4.

Finally, this list of attribute assignments gets transformed into a request made to the Hermes TCP/IP interface, the start of which is shown in figure 7.5.

```
roleActReq(
           _,                               % any source session
           _,                               % any target session
           _,                               % any context
           role(doctor,v1,_,_,[12,_,_],_).  % the credential of interest
           _                                % at any time
          )
```

Figure 7.3: A Prolog term matching an example role activation request

```
CredPat_f       = "role"
CredPat_f_n     = "6"
CredPat_f_1_n   = "1"
CredPat_f_2_n   = "1"
CredPat_f_3_n   = "1"
CredPat_f_4_n   = "1"
CredPat_f_5_n   = "3"
CredPat_f_5_1_n = "1"
CredPat_f_5_2_n = "1"
CredPat_f_5_3_n = "1"
CredPat_f_6_n   = "1"
CredPat_f_1     = "doctor"
CredPat_f_2     = "v1"
CredPat_f_5_1   = "12"
```

Figure 7.4: A functional term transformed into an atomic term list

```
subscribe.
roleActReq.
7002
CredPat_f.
equals.
role.
CredPat_f_n.
equals.
5.
CredPat_f_1_n.
equals.
1.
CredPat_f_2_n.
equals.
1.
CredPat_f_3_n.
equals.
1.
CredPat_f_4_n.
equals.
1.
CredPat_f_5_n
equals.
3.
and so on...
```

Figure 7.5: The start of an example Hermes TCP/IP subscription request

```
privilege(PrivName, Version, Session, Context, [AttrValue|List])
activate(role(RoleName, Version, Session, Context, [AttrValue|List]))
```

Figure 7.6: EDSAC layer 0 interface predicates

7.2.2 EDSAC layer 0 interface

In our prototype, many of the features required for layer 0 EDSAC functionality are provided by the Prolog engine. For the policy cache, authoritative credential store, and the session database, we use areas of the memory-resident Prolog clause database. In our tests we restricted ourselves to situations in which this credential and session state is memory-bound. It would be a reasonably simple matter to create Prolog clauses that unify with all the credential and session searches, and pass requests and replies to and from a database engine external to Prolog. We felt it was acceptable to assume memory-bound operation, particularly since the event system lets us avoid caching lots of policy state at any one given node.

For rule evaluation we employ Prolog's own inference engine. Compared to the OASIS rule evaluation process, Prolog lacks native type support, and also does not provide guaranteed termination of clause evaluation. As mentioned above, we were able to model the basic effect of a type system using functional terms, although this would not be sufficient for policy validation in commercial deployments. In terms of termination, we compiled our extended OASIS XML policies into Prolog clauses using XSLT. This was possible because Prolog clauses and OASIS rules are quite similar in form. We were thus able to ensure OASIS-equivalent termination properties for them, without precluding the option of alternative rule structures.

Our EDSAC prototype layer 0 policy representation and inference process are closely related to OASIS. The basic predicate templates for invoking OASIS-style privilege authorisation and role activation rules are shown in figure 7.6.

In our OASIS-like policy language the privilege/role name and version terms together identify a set of rules that might be able to reach that target. `actionReq` events trigger a search through the local clause database in order of clause definition for the first clause that can successfully infer the target of the action request. Figure 7.7 shows the clauses we assert and revoke to and from the Prolog database to maintain session state. Role and appointment state is recorded using `role` and `appointment` clauses respectively. The `membershipCond` clauses indicate which currently-active roles would need to be revoked when this node is notified of their prerequisite role being revoked. The `InferenceSource` term will indicate whether any matching clause from the Prolog database is an active role (`fact`), or a rule that might potentially be able to activate the role (`rule`).

```
role(RoleName, PolVersion, Session, [AttrValue|List], InferenceSource)
appointment(ApptName, PolVersion, Session, [AttrValue|List])
membershipCond(PrerequisiteRole, TargetRole)
```

Figure 7.7: EDSAC layer 0 session state

```
actionOnBeat(Heartbeat, Term)
edsacSubscription(Event, Term)
```

Figure 7.8: EDSAC layer 1 session state

7.2.3 EDSAC layer 1 interface

The main addition we need to make to support EDSAC layer 1 requirements is the ability to use the publish/subscribe system to communicate between components. We can reuse the policy and state predicates we defined for layer 0, but must ensure that, where necessary, role activations use the two-phase protocol. For roles that are tagged for dynamic constraint checking, we use XSLT to generate both a request clause and a callback clause. The extra layer 1 session database predicates are shown in figure 7.8. The dynamic clause `actionOnBeat` indicates a term to evaluate at a given network time. We would normally expect this term to be a particular role activation clause. The `edsacSubscription` maintains the dynamic event subscriptions for this node, indicating the `Term` to evaluate on receipt of the given `Event`.

7.2.4 Extensions to OASIS policy used in our prototype

The EDSAC layer 1 interface brings with it the ability to use events in the inferencing behaviour of our policy engine. As we have noted previously, we intend the EDSAC architecture to work with any role-aware policy language. Our familiarity with OASIS made it the obvious choice for testing our prototype EDSAC system. However, membership conditions are the only explicitly event-driven part of OASIS. This section introduces the extensions to the OASIS language we use throughout the rest of this chapter to exercise EDSAC-specific features. We will refer to this augmented language as the EDSAC policy language, without intending this to be exclusive of other potential policy languages in future.

Membership conditions

The `membership` attribute was introduced to the CBCL OASIS policy language as an optional tag for role activation rule prerequisites. It indicates that prerequisites with this attribute must remain valid for the target role to remain valid. This concept has been present since early OASIS research, but dynamic role behaviour has never really been tested in OASIS implementations, and was thus missing from the CBCL OASIS policy language. It is now present in both the EDSAC policy language and the EDSAC prototype implementation. For example, the following subsection of EDSAC policy indicates that the `aeTriageNurse` role (i.e. accident and emergency triage nurse) may only remain active while the `nurseOnDuty` role with matching *id* attribute remains active.

```
<activation role="aeTriageNurse" context="aeward1">
  <condition name="nurseOnDuty" membership="yes">
    <with name="id" value="$id" />
  </condition>
</activation>
```

Exclusive role activation

The `exclusive` attribute on an activation rule adds a reactive constraint to the specified target role. If an exclusive role is active on an EDSAC node, this node will reject other principals trying to activate a similar role within the same policy context. Exclusive roles always use the two-phase role activation protocol, and provide the simplest form of dynamic constraint. For more power, constraints on the policy contexts themselves need to be specified (§7.2.4).

We can indicate that only one triage nurse should be active within the context `aeward1` by changing the first line of our `aeTriageNurse` example above to:

```
<activation role="aeTriageNurse" context="aeward1" exclusive="yes">
```

This leads to assertion of an EDSAC event subscription being generated by our XML policy to Prolog XSLT transformation. The resulting Prolog clause is shown in figure 7.9 (with whitespace added for readability).

Role delegation

An addition to the syntax for authorisation rules is the `delegate` tag. This excerpt from our workflow case study (§7.4.3) shows how the tag is used in policy files:

```
<authorization privilege="delegateCheckVendorAccount">
  <authorizing-role name="wf_checkVendorAccount">
    <with name="wfid" value="$wfid"/>
    <with name="wfoid" value="$wfoid"/>
  </authorizing-role>
  <delegate />
</authorization>
```

```
edsacSubscription(
  roleActReq(
    _,
    _,
    aeward1,
    role(aeTriageNurse,v1,_,aeward1,[Vid],fact),
    TargetBeat),
    (
      role(aeTriageNurse,v1,_,_,_,fact),
      edsacSend(
        roleActVote(
          _,
          _,
          aeward1,
          role(aeTriageNurse,v1,_,aeward1,[Vid],fact),
          -1
        )
      )
    )
  ).
```

Figure 7.9: A subscription enforcing exclusive role activation

As for OASIS, the EDSAC policy language follows the principle of only allowing one role to be a prerequisite in an authorisation rule. The `delegate` tag indicates that successful acquisition of the target privilege will be accompanied by an EDSAC event forwarding the prerequisite role to some set of target sessions. This has the effect of delegating the role to other principals.

Note that the delegated role will be activated in its own right within the specified context at the destination node(s). This destination role activation will use the two-phase role activation protocol if necessary. This means that delegation cannot circumvent context constraints. The example provided in section 7.4.3 demonstrates EDSAC role delegation.

Context-driven constraints

Policy contexts were introduced in section 4.4.2 for the sake of policy administration and information flow analysis. As mentioned in section 6.4.2, in our EDSAC prototype contexts provide the role containers in which we manage dynamic constraints. These contexts allow us to target dynamic constraints at certain groups of active roles. Note that the active roles in any one context may come from any number of different user sessions.

In our EDSAC implementation contexts appear in two places. First, the definition section of a policy may contain them. We include two example context definitions from the workflow case study presented in section 7.4.3:

```
<context name="wfc_checkData">
  <cardinality max="1" />
</context>

<context name="wfc_checkOrderType">
  <latch>
    <role name="wf_checkVendorAccount"/>
    <role name="wf_checkCredit"/>
  </latch>
</context>
```

The `cardinality` node defines the maximum number of roles that can be active in a given context. Note that a particular role might well be activated into many different contexts. In the EDSAC policy language, the optional `context` attribute of role activation rules determines whether this role participates in context-controlled dynamic constraints. We expect cardinality context constraints to have widespread applicability. The latch type of context constraint presented is likely to be useful mainly for workflow management.

The `latch` node indicates that after a role within the latch set has been activated in a context, only further instances of that particular role can subsequently be activated in that context. We use these contexts in our EDSAC implementation to ensure that teams of principals make consistent task choices within workflow environments. The term ‘latch’ is used in digital electronics as a switch which can be left in a given number of states – our use of the term is comparable if switch states each select a given role.

Note that EDSAC nodes that activate roles into a given context need not contain the event subscriptions that check dynamic constraints. In our implementation only the policy files that define constraint properties result in nodes that check these properties. It is an EDSAC layer 2 design decision as to whether some particular nodes are designated as being responsible for managing contexts. In our workflow case study, most of the context constraints were managed by one particular node. Multiple nodes managing the same contexts would cause redundant context state storage. Whilst this may be desirable in some systems, in our case study it would have further complicated the diagnostic output. It would also require that these context-managing nodes agree how they will resynchronise their state should any of them become disconnected. The cardinality constraint above will be transformed via XSLT into the Prolog predicates presented in figure 7.10 (reformatted for readability).

The first of these clauses ensures that role activation requests are rejected when this context is 'full'. The second and third event subscriptions update the number of roles counted in this context in reaction to activations and revocations respectively.

In addition to reacting to role activation requests based on context constraints, EDSAC nodes can also explicitly delegate and revoke prerequisites on other EDSAC nodes. It is thus possible to design policy rules that require delegations to activate and lever control over remote role activations this way too.

7.2.5 Secure access to term-servers

The EDSAC interface presented so far does not contain an authentication manager. Also, because our Prolog processes only use a single thread, there is a risk that connection buffering relies too heavily on the socket implementation. We address these concerns by implementing a multi-threaded communication manager in Java. Each Java process presents a secure TLS [Die99] socket interface to the network, and provides a request-queueing mechanism. For simplicity, the internal communication between these Java processes and the Prolog term-servers they protect still uses local sockets, but this could easily be replaced by other methods of inter-process communication.

A scalable manner in which to provide such Java network services is through Java Servlets [Sun96]. Whilst Servlets can be used for any form of communication, they are often employed to provide dynamic web-content, alongside such dynamic document-oriented technologies as Java Server Pages (JSP) [Sun99]. Older Common Gateway Interface (CGI) techniques for dynamic document generation require the web server to spawn new processes to handle each request. A servlet container uses light-weight threads to service each request instead.

During the development of EDSAC it became clear that the Apache project's open-source Tomcat servlet container was far simpler to use than the Orion J2EE application server, and had much higher quality documentation.

```
edsacSubscription(
  roleActReq(_,_,
    wfc_checkData,
    role(R,V,S,C,A,fact),_
  ),(
    contextFull(wfc_checkData),
    edsacSend(
      roleActVote(_,_,
        wfc_checkData,
        role(R,V,S,C,A,fact),
        -1
      )
    )
  )
).

edsacSubscription(
  roleActConfirm(_,_,
    wfc_checkData,
    role(_,_,_,_,_,_)
  ),
  contextUpdate(wfc_checkData,1)
).

edsacSubscription(
  credRevoke(_,_,
    wfc_checkData,
    role(_,_,_,_,_,_)
  ),
  contextUpdate(wfc_checkData,-1)
).
```

Figure 7.10: A subscription enforcing a cardinality constraint

TLS/SSL secured EDSAC interface

Our secure servlet interface to EDSAC requires all connections to use client-side certificate-based authentication in addition to the more conventional server-side authentication. At least one of the certificates on the client-side certificate chain will need to be contained within the servlet's trust repository. In cases where full authentication is not required, such as in trust-based access control, we rely on other components (e.g. a trust engine) to authenticate on behalf of the principals for which they are willing to vouch.

Client-side authentication provides the servlet with an appointment certificate with which to begin EDSAC operations. After a connection is established, we provide a very simple text-based protocol through which to proxy requests to the underlying EDSAC term-server. The command strings and their functions are described in table 7.2. They are parsed one per line; multiple requests can be issued one after the other except for the `edsacCommand` and `adminCommand` strings, which both forward the following line until the end of the stream to the underlying term-server.

Note that all of the events shown in figure 7.2 already contain a session identifier as their second argument. We maintain the session identifier argument to the `edsacCommand` string for future implementations in which either the EDSAC commands can be generated to contain the valid session identifier or at least those submitted are checked for correctness.

Every use of the `addAppointment` command (which may be an implicit result of using the `requestSession` command) asserts the current client-side certificate's EDSAC X.509 extension fields into the indicated session on the Prolog term-server. The reason that the client-side certificates are not linked one-to-one with the session identifiers is so that a given principal may assert multiple appointment certificates into their current session. We felt that this approach allows more flexible policy design than the CBCL OASIS implementation in which there was such a one-to-one certificate-to-session mapping.

Unlike generic TCP/IP sockets, starting TLS or SSL sessions is a highly expensive operation. This is due to the certificate validation requiring the use of asymmetric cryptography, rather than the much faster symmetric cryptography employed following a successful handshake and establishment of a shared session key. Consequently we assume that connections made to a particular TLS wrapper for an EDSAC Prolog term-server are likely to be long-lived. Should the connection need to be closed and reopened, SSL and TLS negotiate a session ID that can be used to skip the full handshake protocol on future connections.

7.3 Location-aware access control

This section describes our ongoing research into location-aware access control. By the term 'location-aware' we capture two separate but related technologies. One is access control to location data. The other is location-based access control, i.e. making access control decisions based on the location of a principal. Extending the former into location-based access control is the subject of our ongoing collaborative research.

Command String	**Function**
`requestSession`	No further input is required. A unique EDSAC session identifier string will be generated by the authentication module, and returned as a string terminated with a new line. The `addAppointment` command will also be implicitly executed using this new-session identifier.
`addAppointment` <*session identifier*>	This command asserts the EDSAC/OASIS attributes from the current appointment certificate into the specified session within the EDSAC session database.
`endSession` <*session identifier*>	This command explicitly requests that the session state for the identified session be removed. This will cause revocation of all the roles activated within it.
`edsacCommand` <*session identifier*>	The text lines following this command string will be passed to the term-server for parsing as EDSAC commands within the given session.
`adminCommand` <*session identifier*>	This is a command only available to principals who have added an administrator appointment to the identified session. An administrator privilege check term containing the given session identifier is inserted before the text lines that follow the command string. This term list is then forwarded directly to the term-server underlying this EDSAC node.

Table 7.2: Prototype EDSAC authentication service commands

The increasing prevalence of portable computing equipment has been coupled with developments in location-aware technology. The nature of location data is currently radically different for each class of device. Mobile phone handsets, for example, have always needed to be location-aware in the sense of knowing their nearest cell base-station. The computational capacities of the phones themselves have undergone significant recent developments (they are increasingly converging with Personal Digital Assistant (PDA) functionality). They are now capable of communicating with Internet services using protocols previously too heavyweight for such devices.

At the other end of the spectrum, personal computers increasingly portable. The spread of 802.11b (and to a lesser extent 802.11a and 802.11g) wireless networking, also known as 'Wifi', enables computers to appear on the Internet in an *ad hoc* manner.

Ubiquitous, pervasive computing is usually presented in terms of the extra convenience it offers. However, it also has the potential to be highly intrusive. We feel that the increase in information about where a device is geographically should be coupled with increased control by users over that information.

Our initial location-based access control experiments have been performed at the Laboratory for Communication Engineering (LCE) in Cambridge. The LCE has installed a location-detection system [WJH97, Cou04] in their offices that allows highly accurate positioning of 'Active Bats' carried by people and attached to devices. The accuracy is often within a few cubic centimetres (plus orientation can often be inferred), and location information is refreshed every few milliseconds (although the refresh rate is dependent on the number of bats being tracked – currently more than fifty).

Alastair Beresford has researched anonymity within location-aware systems using data from the LCE Active Bat system – for example, under what circumstances it is possible to be sure you get "lost in a crowd" of location-aware [BS03] (related work includes [MFD03]) devices. It was thus natural that we should work together to apply OASIS and EDSAC technology to management of location-aware policy.

7.3.1 Why is location data special?

Many of the concerns relevant to location-aware policy have been highlighted by other researchers [SMB03, SBM03, HS04]. Location information is different from the data handled by traditional access control applications in many ways:

Variable precision. Location information can easily be handled in a non-binary manner. While a request to read a file on disk will either be accepted or rejected, location information can have its precision reduced, and still be useful. For example, a mobile-phone owner may be happy for family members to be provided with the phone's location to within one hundred square metre accuracy. Their employers however, may only be able to gather coarse location information – perhaps merely identifying which city the user is in.

Peer to peer queries. Many access control scenarios operate in a client/server manner. Users make a request of a centralised resource. Decisions relating to location are much more likely to be made based on the current context of both requester and requestee. Contrast a user making a request of a central EHR database with the situation in which one employee is querying the whereabouts of their peer. In the latter case, a precise response might be issued only if both parties were active in a given employee role.

Highly dynamic. Location data will, by the very nature of mobility, change rapidly. It is thus important that any location-aware access control system is able to manage such changes efficiently.

Distribution. If policy evaluation occurs on the actual agents moving throughout a network, support will clearly be required for distributed policy evaluation protocols.

User policy. In more traditional access control environments, the policy relating to resource access will usually be dictated by the management of the organisation to whom those resources belong. Users will implicitly or explicitly accept these rules in the process of becoming part of that organisation. Location information may have significant personal implications, however. We believe end users are likely to want to have a range of location-disclosure policies available to them from their organisation. Moreover, these users are going to want to adjust the access control policy that relates to their location themselves.

We believe the EDSAC architecture can address these concerns, as discussed below.

7.3.2 EDSAC and location-aware access control

We were hopeful that roles (and RBAC) would be able to assist in both the types of location-aware access control decisions outlined above. We found that the NIST RBAC sense of roles did not meet the requirements of location-aware systems well enough. We needed more general notions of credentials such as those found in EDSAC. The primary difference comes from the dynamic and peer-oriented nature of location information mentioned above. In traditional RBAC, the notion of role activation is usually tied to specific user requests (or arises from static relationships). Roles are often classified as either functional or organisational, yet both of these classifications link a principal into an overarching organisational structure. This is inappropriate for users controlling access to their own location information. Our experiments indicate the need for some sort of *situational role* or *environmental role* (see [CLS$^+$01]) classification to cater for non-organisational concepts such as “being in meeting room 3”, or a role `location` that contains attributes indicating position and precision. In other words, we need dynamic context-awareness.

The EDSAC role activation protocol supports the dynamic communication of changes in credential status very well. The publish/subscribe basis of EDSAC communication means that participants in an access control network can easily limit their ‘interest’ to events that directly pertain to them. For example, office resource management software can subscribe to situations in which principals enter a meeting room, and schedule future *ad hoc* meetings to other rooms.

To see how the EDSAC events facilitate variable precision in the disclosure of location information, consider an office system in which users have a situational role `location`. Say this role has parameters indicating what *floor* an employee is on in an office building, as well as the *office* area they are within on that given floor. The location system in the office does not need to use the two-phase role activation protocol to update this role credential, since this credential is owned exclusively by the principal. Of course if a principal is at a certain location, this may cause dynamic constraints to come into effect when they try to gain other privileges (e.g. trying to redirect their internal phone routing to an office in another city).

The location management system updates the employee's `location` credential using credential forward and credential revocation events (§6.2.2). However, either using attribute encryption or independent location update events, we can separate nodes that only gain information about what *floor* the employee is on from those who may also see which *office* the employee is using. Controlling the subscriptions to the `location` events is effected through EDSAC layer 1 policy (i.e. which rules are permitted to contain a `location` prerequisite including the *office* attribute).

The EDSAC architecture aims to facilitate peer-to-peer queries by allowing a range of different types of EDSAC nodes on an access control network – at layer 1 we do not require specification of which nodes are key to the operation of a given distributed system since the publish/subscribe framework should provide appropriate communication spanning trees regardless. Our Prolog prototype also allows very lightweight nodes to be built (assuming sufficient link-level security can be achieved via other means).

7.3.3 Test environment

Within the LCE offices, each office phone has a touch screen that provides a graphical thin-client (using the Virtual Network Computing (VNC) protocol [RSFWH98]) to various useful services. One such service is an office map that indicates the location of each Active Bat within the current view.

Our aim was to engineer access control policy into this system, so that the requests for location updates are at least partly under the control of the users they affect.

Rather than designing a system in which a centralised policy engine would determine the results for each location request, we were keen that we leave the possibility open for the actual portable devices to control their own disclosure policy. This has the advantage that the user carrying a given location-aware device can have direct and immediate control over its location-disclosing behaviour. Our current EDSAC Prolog implementation was designed in part so that it could interface easily with a wide variety of such applications. In particular, it is significantly less heavyweight than the CBCL OASIS implementation – we felt the J2EE application server deployment model was not appropriate for the LCE environment.

The Active Bat software ('Spirit') has good interfaces for application development. Requests can be made for raw location events, but, more usefully, call-backs that correlate raw location information with a world model (e.g. what room a person is in) can be registered. We unified our Prolog term-servers with the Python module for the Spirit API.

Our initial tests demonstrated that the Prolog EDSAC code could easily service policy applied to raw location events – in fact, way beyond our expected needs in managing call-back information.

Policy design

There is no point in providing a flexible access control system if it cannot be conveniently configured by users. We believe there will be two fundamental ways in which users can influence the access control policy relating to their location. At the lowest level, users could author specific policy rules in the appropriate language for the EDSAC node representing them. More commonly, users will interact with policy in a more controlled manner – policy designers will build tunable attributes into a given set of policy rules. After all, it is unlikely that most non-technologist users will want to interact with the policy language directly. We believe many policy configuration tasks will be sufficiently predictable to enable policy templates to be defined centrally and tuned locally.

For example, in an office scenario, policy relating to the disclosure of user location information may allow these users to specify certain 'dark-zones', in which they do not appear on the general location map. Our aim for the LCE policy management of these dark-zones is to provide a point-and-click interface where location disclosure can be turned on and off for each room on the map. This is an example of high-level policy design – in fact the policy rules at the lower level will not need to be modified if the set of dark-zone rooms changes, since in EDSAC terms an environmental predicate will check whether a given room is contained within the set of permissible rooms for location disclosure. Users requiring more complex policy rules may well be able to write them using the underlying EDSAC policy language directly, or be assisted to do so.

7.3.4 Experiments applying access control to location data

In this section we present a very simple EDSAC example in which we perform access control decisions over location data with dynamic context awareness. This is a necessary precursor to the full location-based access control systems in development.

We summarise the diagnostic output from our software in this section but provide full listings in section A.1. We are in contact with an LCE member who is developing more comprehensive software for their touch-screen phone user interface. When that work is complete we will be in a position to complete our EDSAC integration there.

The structure of our experiment is shown in figure 7.11. Say we have two users, Alice and Bob. The figure models Alice making requests about Bob's location. There are three parties in the EDSAC network; nodes 'A', 'B' and 'C'. Node 'A' is operating on Alice's behalf. Node 'B' contains the policy under Bob's control relating to release of his location. Node 'C' is the system that actually 'knows' about the locations of principals.

Since EDSAC uses content-based delivery of its events, Alice does not need to consider where her requests are being processed. In the example here, given the simplicity of the

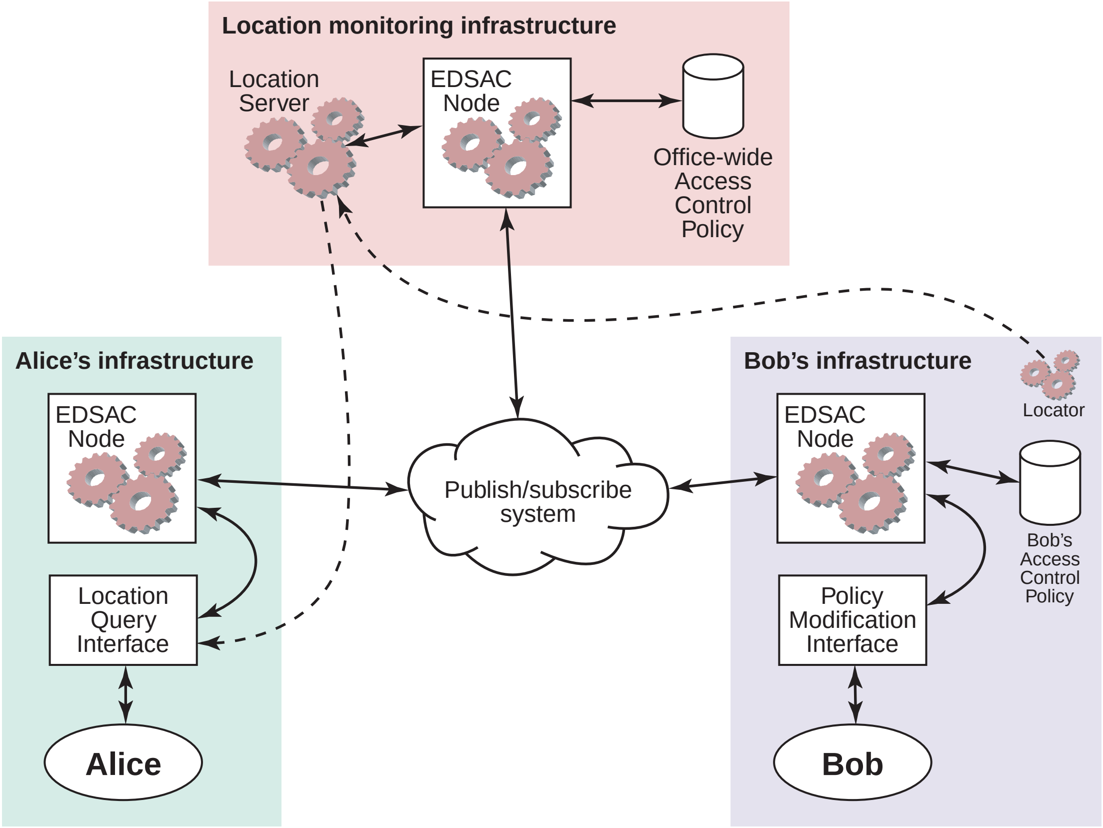


Figure 7.11: Structure of our experiment for access control to location data

policy, we are modelling a situation in which Bob chooses to run his own EDSAC node. Alice thus activates her roles within a user session running on Bob's computer. This allows Bob a great deal of control over the rules that relate to the disclosure of his location. In future it may be possible that the core EDSAC node operates on an even smaller embedded device he carries with him. This would allow him to make changes to his particular location disclosure policy whenever it suited him.

Our example assumes that centralised infrastructure keeps track of Bob's actual location. This is certainly the case in the LCE environment, since a centralised facility needs to monitor all the detectors located in the office ceilings and smooth out any erroneous readings.

Of course a different EDSAC layer 2 design could choose to centralise all or some of the policy running on Bob's equipment to a shared server (e.g. if an employer wanted to enforce certain permanent disclosure rules), or to distribute the location-measuring nodes. PDAs with GPS receivers could potentially perform the roles of both nodes 'B' and 'C' in our experiment.

Table 7.3 describes the events that Alice causes to be sent from her EDSAC node. All of her events are directed at node 'B'. Alice requests both the `coarseLocation` and the `fineLocation` privilege three times, first before activating any roles, then after activating

Time	Action
3	print active roles
3	request privilege `coarseLocation`
3	request privilege `fineLocation`
4	activate role `generalLocator`
4	print active roles
4	request privilege `coarseLocation`
4	request privilege `fineLocation`
5	activate role `knownLocator`
5	print active roles
5	request privilege `coarseLocation`
5	request privilege `fineLocation`

Table 7.3: Actions Alice requests of Bob's location disclosure agent

the role `generalLocator`, then after activating the role `knownLocator`. We have also interspersed commands to print the currently active roles at node 'B'.

The output in figure 7.12 shows the events flowing in and out of node 'B'. For all the abbreviated diagnostic output presented in this chapter the first seven characters of each line indicate the overall class and timing of each message. Times are provided as heartbeat serial numbers in square brackets. There are five possibilities for the first three characters of a line. An event being published is indicated with '`<--`', and an event being received due to a subscription at this node with '`-->`'. Commands directly executed on this node begin '`:::`' or are blank to indicate continuation of the previous line's output. Finally, the line prefix '`---`' indicates the callback from a successful two-phase role activation. In terms of event attributes, our abbreviated syntax uses '`r`' for role, '`s`' for source session, '`c`' for context, '`hb`' for heart beat, '`p`' for privilege, '`t`' for target session, and '`w`' for vote weight.

All incoming requests are from Alice (i.e. node 'A'). The outgoing events primarily target node 'C' (the privilege vote events), although there are also a number of events caused by standard EDSAC role activation behaviour.

Finally figure 7.13 shows the output that is generated by the EDSAC node that actually 'knows' Bob's location (node 'C'). As expected we see no output from when Alice made privilege requests without activating roles. We see the `coarseLocation` privilege being granted when Alice has activated the `generalLocator` role (lines 2 and 4). This role indicates a principal with a right to know basic location information. When Alice further activates the `knownLocator` role, Bob's access control policy recognises that Alice should be permitted to know his location in detail, hence the `fineLocation` privilege is also granted to her (line 6).

```
:::[3] Active roles in this database: []
-->[3] actionReq      p=coarseLocation a=[1]
-->[3] actionReq      p=fineLocation   a=[1]
-->[4] actionReq      r=generalLocator
<--[4] roleActConfirm r=generalLocator
:::[4] Active roles in this database: [generalLocator]
-->[4] actionReq      p=coarseLocation a=[1]
<--[4] privVote       p=coarseLocation s=9000 w=1
-->[4] actionReq      p=fineLocation   a=[1]
-->[5] actionReq      r=knownLocator
<--[5] roleActConfirm r=knownLocator
:::[5] Active roles in this database: [generalLocator, knownLocator]
-->[5] actionReq      p=coarseLocation a=[1]
<--[5] privVote       p=coarseLocation t=9000 w=1
-->[5] actionReq      p=fineLocation   a=[1]
<--[5] privVote       p=fineLocation   t=9000 w=1
```

Figure 7.12: Diagnostic output from our policy decision point

```
-->[4] privVote p=coarseLocation t=9000 w=1
:::[4] Location of principal <1> is: <151.191, -33.8664> with error 1.
-->[5] privVote p=coarseLocation t=9000 w=1
:::[5] Location of principal <1> is: <151.191, -33.8664> with error 1.
-->[5] privVote p=fineLocation   t=9000 w=1
:::[5] Location of principal <1> is: <151.165, -33.9379> with error 5e-05.
```

Figure 7.13: Diagnostic output from the location monitor

7.4 Workflow and dynamic privilege management

This section discusses workflow management systems. Workflow introduces considerations beyond many traditional access control systems. The output from our test case (§7.4.3) demonstrates EDSAC managing dynamic constraints, fast role revocation and credential delegation to achieve workflow progression in a distributed environment. We provide our definition of the term 'workflow':

> "Workflow management involves technology controlling and monitoring business processes in an organisation. Workflows specify the valid progression of work units between different teams and individuals."

At its most basic level, workflow management simply ensures that a given task has all of its sub-tasks performed in a valid order, and that all required sub-tasks have indeed been satisfactorily completed. An example of a workflow might involve publishing pages on a website. Any registered user might be allowed to author content, but the workflow requires that this content is reviewed and approved by two other users before it becomes 'live'.

It is usually permissible for tasks to be achieved in a flexible order. Often workflows will involve teams of principals; managing parallelism becomes important. We examine workflow systems because they provide a rich source of rapidly-changing dynamic constraints – precisely the type of application that suits the EDSAC architecture.

Unlike access control over particular resources, the specification of a workflow is general over all instances of the tasks to which it refers. This means that the policy specifying workflow for publishing a given type of document within an organisation, for instance, will apply as a template over each specific document being created.

Many researchers have already proposed models and policy languages that integrate workflow monitors into access control systems (see [ACM01, WBK01, WKB01, Tom99, KRSR98, BFA97, vdA96, KCK01]). In our test cases we have used an OASIS-like policy language augmented with dynamic policy contexts to guide workflow progression. The unique aspect of our work is that we achieve workflow-style constraints in a distributed system through content-based delivery of EDSAC events and the two-phase role activation protocol rather than adding workflow-specific extensions to our policy language. Other policy languages could be used on any of our EDSAC nodes, provided that they can support a two-phase role activation protocol.

Schemes for combining access control and workflow presented previously have generally relied on a centralised workflow monitor. Although our example uses a much more distributed approach, we could also build systems that check workflow state centrally – the EDSAC layer 2 design specifies whether particular EDSAC nodes have a specific function in a distributed environment. Scaling up interacting EDSAC workflow networks into federations would involve layer 3 design decisions. The CrossFlow project [KGV99] has focused on research into cross-organisational workflow support.

Another difference between workflow systems and traditional access control is that the latter is normally reactive, whereas the former is proactive. By this we mean that access control systems normally either grant or deny a request, but that request is made directly by some principal. In workflow management systems, particularly when deadlines are being managed, the user interface of the workflow management system will provide the user with information about the state of their outstanding tasks. We discuss our support for deadlines and obligations in section 7.4.4, although do not focus on the user interface requirements of EDSAC-supported workflow systems.

7.4.1 Workflow specification

To ensure that our workflow scenario was realistic, we examined the material made publicly available by the Workflow Management Coalition (WfMC) [WfM95]. The WfMC is a large Internet consortium that provides guidance in the design and implementation of workflow applications. A number of workflow specification languages have been proposed by the WfMC, however we felt the low-level XML Process Description Language (XPDL) best matched normal EDSAC privilege granularity.

We downloaded the sample workflow process XPDL file available from the WfMC website and wrote XSLT to transform it from XPDL to Dot for conversion to EPS (a similar process to that discussed in section 5.1.3). The output of this transformation is shown in figure 7.14.

A significant amount of the XPDL language is devoted to managing attributes being passed between 'participant' processes. This is as a consequence of workflow moving toward specification of web-service federation, instead of describing exclusively human processes. Our interest was on how workflow progression changed the set of user privileges available rather than on the data being passed through a system. As a consequence we were able to dramatically simplify the workflow specification we used as the basis for generating EDSAC policy rules.

Workflow graphs comprise of nodes that represent particular tasks (or sub-workflows) and connections between these nodes that indicate conditions on workflow progression. We represent the individual tasks in a workflow graph via symbols, a, b, and so forth. Sets of tasks are written T, S, etc. The workflow graph may be represented using the following relations:

$$\begin{gathered}\mathbf{split}(a, T, type, conditions)\\ \mathbf{join}(S, b, conditions)\\ \mathbf{step}(a, b, conditions)\end{gathered}$$

The **step** relation indicates the set of *conditions* that needs to be satisfied in order for a principal to be permitted to progress from task a to task b. The *conditions* parameters in the **split** and **join** relations have the same meaning. The *conditions* must be able to be expressed as an ordered conjunction of predicates and prerequisites if they are to be encoded into the EDSAC policy we use in our workflow demonstration.

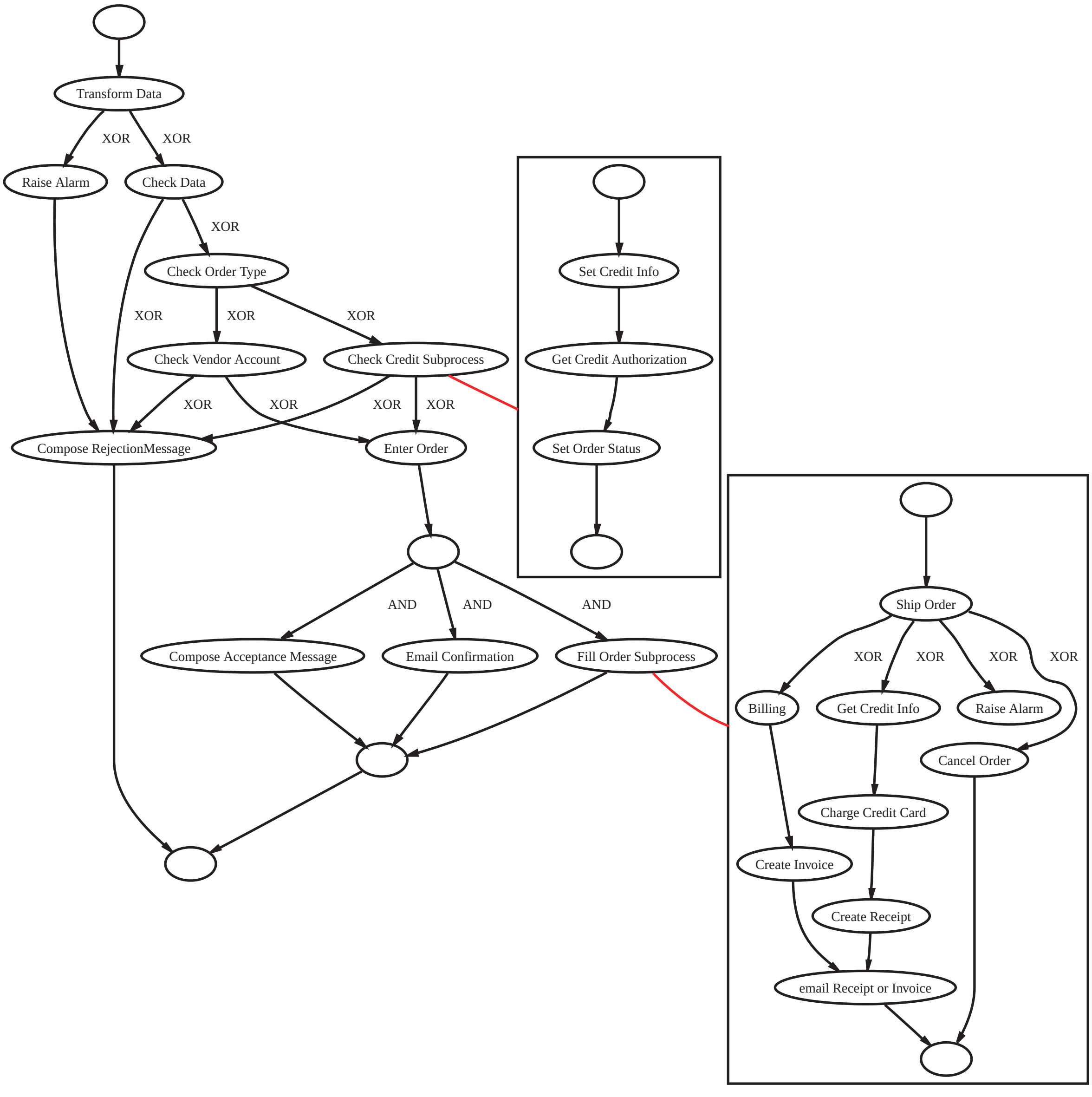


Figure 7.14: Our example workflow

The **split** relation indicates a point in the workflow where participants have a choice of path. The starting points for each possible path are contained in the set T. The *type* of split will either be `AND`, or `XOR`, as specified in the XPDL language, and displayed on the transitions in figure 7.14. The **join** indicates that all the tasks in S each proceed to the task b. XPDL graphs are in fact lattices, thus there is no need to specify a *type* for **join**s, since this can be inferred from the **split**s above it.

An `AND` transition indicates, in the case of a split, that all outgoing paths must eventually be completed, but that their execution can occur in parallel. The `XOR` (exclusive or) transition means that only one possible outgoing path is permissible in a split.

7.4.2 Mapping workflow entities into policy

The transformation from workflow into access control policy rules must take into account both the connection between workflow tasks, and the type of any splits involved. Further, since all workflow policy rules will apply independently to each workflow instance, the policy rules need to be parameterised with a workflow instance identifier.

The task nodes in our workflow lattice are translated into *workflow-roles* and the edges into rules in our policy language. Consider first a lattice in which all splitting points are of type `XOR`. In such cases, every edge in the lattice will lead from some workflow-role r to a workflow-role t. Each such edge will generate a policy rule with target t that has r and the transition *conditions* as rule prerequisites. When we include splits of type `AND`, eventually some policy rule must recombine the different paths by including multiple workflow-role prerequisites. An example of such a situation is presented in figure 7.15. Note that the rule that combines the paths through multiple workflow-role prerequisites must have an unambiguous derivation. The example shown in figure 7.16 has an `AND` split, one branch of which leads back to the other via an `XOR` split. Disambiguating this situation involves adding further role prerequisites.

We have not examined the sub-flows in figure 7.14, but the top-level workflow is translated into the role structure shown in table 7.4. See figure 7.17 for the rules that appropriately connect the workflow-roles together in our example. Note that all workflow-roles are parameterised with both a *wfid* and a *wfoid* to indicate the workflow identifier and the identifier for this particular instance of the workflow respectively.

Until an entire workflow is complete all prior roles in that workflow instance remain active. This does not violate the principle of least privilege as much as it might first appear; policy authors should require the activation of access control roles predicated on workflow-roles, rather than attaching privileges directly to the workflow-roles. We followed this principle in the design of our example in section 7.4.3.

The second conversion involves creating dynamic constraints to prevent invalid workflow paths from being followed. In our example, we use policy contexts to enforce these dynamic constraints. The contexts involved in our example workflow are listed in figure 7.18.

We discussed the two types of context constraint we have used in section 7.2.4. To demonstrate latch constraints, in our example workflow the “Check Vendor Account” task is activated within the `wfc_checkOrderType` context. This means that another team mem-

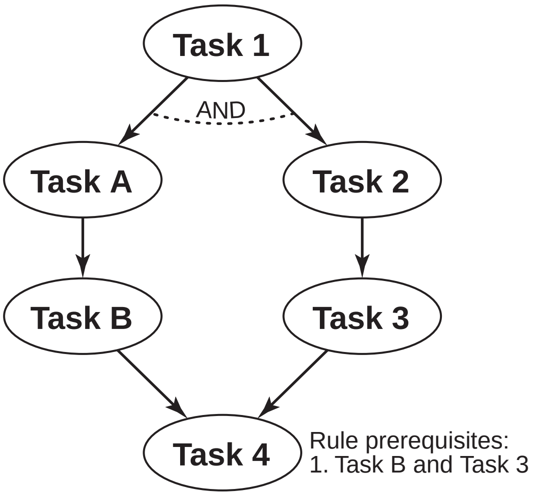


Figure 7.15: A simple example of AND splits generating policy rules

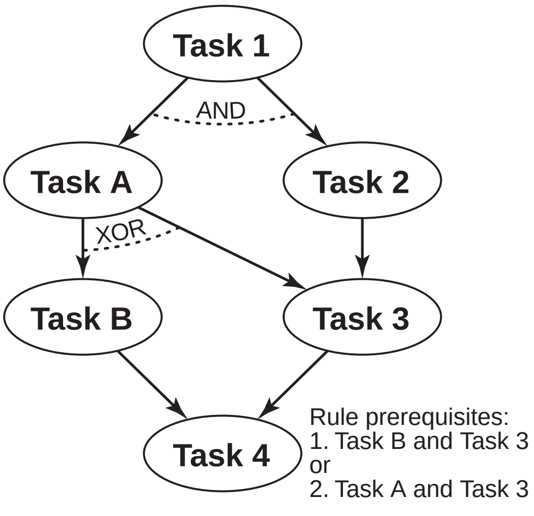


Figure 7.16: Interacting AND and XOR splits in policy rules

$$\begin{aligned}
&\vdash \text{wf\_transformData}\\
\text{wf\_transformData}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_raiseAlarm}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_transformData}(\mathit{wfid}, \mathit{wfoid}),&\\
\text{emulateConditions}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_checkData}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_checkData}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_composeRejectMsg}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_checkData}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_checkOrderType}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_checkOrderType}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_checkVendorAccount}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_checkOrderType}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_checkCredit}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_checkVendorAccount}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_composeRejectMsg}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_checkVendorAccount}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_enterOrder}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_checkCredit}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_composeRejectMsg}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_checkCredit}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_enterOrder}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_enterOrder}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_composeAcceptance}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_enterOrder}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_emailConfirmation}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_enterOrder}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_fillOrder}(\mathit{wfid}, \mathit{wfoid})\\
\text{wf\_composeAcceptance}(\mathit{wfid}, \mathit{wfoid}),&\\
\text{wf\_emailConfirmation}(\mathit{wfid}, \mathit{wfoid}),&\\
\text{wf\_fillOrder}(\mathit{wfid}, \mathit{wfoid}) &\vdash \text{wf\_cleanUp}(\mathit{wfid}, \mathit{wfoid})
\end{aligned}$$

Figure 7.17: The rules that represent our workflow

Node name on workflow graph	Symbolic workflow-role name
Transform Data	wf_transformData
Raise Alarm	wf_raiseAlarm
Check Data	wf_checkData
Compose Rejection Message	wf_composeRejectMsg
Check Order Type	wf_checkOrderType
Check Vendor Account	wf_checkVendorAccount
Check Credit Subprocess	wf_checkCredit
Enter Order	wf_enterOrder
Compose Acceptance Message	wf_composeAcceptance
Email Confirmation	wf_emailConfirmation
Fill Order Subprocess	wf_fillOrder
(final node)	wf_cleanUp

Table 7.4: Workflow-roles in our example

```
<context name="wfc_transformData" >
  <cardinality max="1" />
</context>
<context name="wfc_checkData">
  <cardinality max="1" />
</context>
<context name="wfc_checkOrderType">
  <latch>
    <role name="wf_checkVendorAccount"/>
    <role name="wf_checkCredit"/>
  </latch>
</context>
<context name="wfc_checkVendorAccount">
  <latch>
    <role name="wf_composeRejectMsg"/>
    <role name="wf_enterOrder"/>
  </latch>
</context>
<context name="wfc_checkCredit">
  <latch>
    <role name="wf_composeRejectMsg"/>
    <role name="wf_enterOrder"/>
  </latch>
</context>
```

Figure 7.18: Contexts used in our example workflow

ber cannot subsequently activate the “Check Credit Subprocess” task and violate the XOR split under the “Check Order Type” (see figure 7.14). The option to unwind the workflow so that another path can be chosen would involve privileges that revoked workflow roles and reset the context latch state.

7.4.3 Evaluating our EDSAC workflow implementation

This section evaluates our EDSAC prototype by tracing a given instance of the workflow presented in figure 7.14. We point out salient features of the diagnostic output generated by our EDSAC implementation.

In terms of EDSAC layer 2 design, we chose a deployment structure for our workflow example that involves three EDSAC nodes: a context manager (node ‘C’), and two user controlled nodes ‘A’ and ‘B’. Strictly speaking the EDSAC nodes ‘A’ and ‘B’ are separate logical entities from the users on whose behalf the nodes make requests. However, for simplicity we will refer to the users connected to nodes ‘A’ and ‘B’ as users ‘A’ and ‘B’.

We fully acknowledge that we cannot present an example involving the interaction of three nodes and expect the reader to conclude that our approach is scalable. Our example focuses on demonstrating the semantics of the protocols involved, rather than showing that our implementation itself is capable of managing thousands of nodes.

We do believe our approach is scalable, however. In defence of this claim, we have already discussed the notion of heartbeat synchronisation and network federation at length (see sections 6.3.3, 6.3.4, 6.4.4 and 6.6). We feel justified in assuming that the event delivery mechanism is scalable. Further, the subsections of nodes interested in each others’ events will be partitioned on content-based properties (e.g. no one node should ever need to subscribe to all role events in all sessions and contexts).

One area where we may have scalability problems is in the aggregation of votes on role activations. EDSAC layer 2 design decisions should designate certain nodes to be responsible for certain dynamic constraints (thus limiting the number of interested parties who will vote on role activation events). In terms of the context-based approach we use with our EDSAC policy language, it is a trivial matter to distribute context management to different nodes for the sake of load-balancing, and indeed to leave some redundancy to ensure that no individual node failure will prevent dynamic constraint restrictions.

Our implementation processes each offered a TCP/IP port to the EDSAC network. In the diagnostic output, ports 9000, 9001 and 9016 correspond to node ‘A’, node ‘B’ and the context manager (node ‘C’) respectively.

For full listings of the raw diagnostic output and the complete set of commands issued to the EDSAC nodes, see appendix A. We present an abbreviated form of the commands and node responses below. Even the abbreviated form has a significant amount of detail, so we have divided its presentation into separate figures, and our analysis into three subsections. The sections involve user ‘A’ first trying to achieve all the tasks in a workflow instance themselves, then user ‘A’ enlisting the support of user ‘B’, and a final stage of the two users cooperating to complete the remaining workflow tasks.

Time	Target	Action
1	C	list full contexts
1	C	list latched contexts
1	all	print active roles
1	A	activate role `wf_transformData` in context `default`
1	A	can we enter role `wf_checkData` in context `wfc_transformData`?
1	A	activate role `wf_checkData` in context `wfc_transformData`
6	C	list full contexts
6	all	print active roles
6	A	assert `emulateConditions` predicate to true
6	A	activate role `wf_checkData` in context `wfc_transformData`
11	C	list full contexts
11	all	print active roles
11	A	activate role `wf_checkOrderType` in context `wfc_checkData`
16	A	activate role `wf_checkVendorAccount` in context `wfc_checkOrderType`

Table 7.5: Events in stage one of our workflow scenario

User 'A' working alone

We begin our example with user 'A' completing workflow tasks on their own. In our test cases the initial workflow role `wf_transformData` can be acquired without any prerequisites. A real deployment would treat the creation of a workflow instance as being a privileged operation.

The commands issued to the nodes at different times are shown in table 7.5. The time column of this table refers to publish/subscribe heartbeat serial numbers. The target indicates whether our command was issued to node 'A', 'B', 'C', or all the nodes at once. The action column describes the command that was issued. Tables 7.6 and 7.7 have the same structure.

Abbreviated diagnostic output from the context manager is shown in figure 7.19. Note that the context manager does not play an active part in the first stage of our workflow tasks – it publishes no events. However lines 10 to 13 indicate that it is monitoring activations occurring in some contexts. Were context constraints to be violated by any of the role activation requests, this node would publish negative role activation vote events.

Most of the commands in table 7.5 are directed at node 'A'. Its diagnostic output for this stage is shown in figure 7.20. In line 2 we see the arrival at node 'A' of our request for it to activate role `wf_transformData`. As the root workflow-role, this activation begins an instance of our workflow. Because there are no prerequisite conditions for activating role `wf_transformData`, and it does not need two-phase activation (it needs no dynamic constraints checked), the role activation confirm message is dispatched immediately (line 3).

Having begun a workflow instance, we next request that user 'A' determines whether

```
::: [1] Full contexts: []
::: [1] Context latches: []
::: [1] Active roles in this database: []
::: [6] Full contexts: []
::: [6] Active roles in this database: []
--> [6] roleActReq     r=wf_checkData          s=9000 c=wfc_transformData  hb=10
-->[10] roleActConfirm r=wf_checkData          s=9000 c=wfc_transformData
:::[11] Full contexts: [wfc_transformData]
:::[11] Active roles in this database: []
-->[11] roleActReq     r=wf_checkOrderType     s=9000 c=wfc_checkData      hb=15
-->[15] roleActConfirm r=wf_checkOrderType     s=9000 c=wfc_checkData
-->[16] roleActReq     r=wf_checkVendorAccount s=9000 c=wfc_checkOrderType hb=20
-->[20] roleActConfirm r=wf_checkVendorAccount s=9000 c=wfc_checkOrderType
```

Figure 7.19: Diagnostic output from the context manager during stage one

```
::: [1] Active roles in this database: []
--> [1] actionReq      r=wf_transformData      s=9000 c=default
<-- [1] roleActConfirm r=wf_transformData      s=9000 c=default
::: [1] cannot reach role wf_checkData in context wfc_transformData
--> [1] actionReq      r=wf_checkData          s=9000 c=wfc_transformData
::: [6] Active roles in this database:
        + role=wf_transformData  context=default
--> [6] actionReq      r=wf_checkData          s=9000 c=wfc_transformData
<-- [6] roleActReq     r=wf_checkData          s=9000 c=wfc_transformData  hb=10
---[10] activate:      r=wf_checkData                 c=wfc_transformData
<--[10] roleActConfirm r=wf_checkData          s=9000 c=wfc_transformData
:::[11] Active roles in this database:
        + role=wf_transformData  context=default
        + role=wf_checkData      context=wfc_transformData
-->[11] actionReq      r=wf_checkOrderType     s=9000 c=wfc_checkData
<--[11] roleActReq     r=wf_checkOrderType     s=9000 c=wfc_checkData      hb=15
---[15] activate:      r=wf_checkOrderType            c=wfc_checkData
<--[15] roleActConfirm r=wf_checkOrderType     s=9000 c=wfc_checkData
-->[16] actionReq      r=wf_checkVendorAccount s=9000 c=wfc_checkOrderType
<--[16] roleActReq     r=wf_checkVendorAccount s=9000 c=wfc_checkOrderType hb=20
---[20] activate:      r=wf_checkVendorAccount        c=wfc_checkOrderType
<--[20] roleActConfirm r=wf_checkVendorAccount s=9000 c=wfc_checkOrderType
```

Figure 7.20: Diagnostic output from node ‘A’ during stage one

```
::: [1] Active roles in this database: []
::: [6] Active roles in this database: []
:::[11] Active roles in this database: []
```

Figure 7.21: Diagnostic output from node 'B' during stage one

they can reach the next task, the wf_CheckData role, using their current credentials. Node 'A' responds that it cannot yet reach this role in line 4. To emphasise this point, we cause node 'A' to make an activation request that we know will fail. In line 6 the role is missing as expected.

Here we are simulating that user 'A' has not yet achieved all the necessary conditions required for them to progress to the next workflow task. Checking for completion of the work in a given task is achieved through the *conditions* discussed in section 7.4.2. In our case we simulate the effect of such job completion checks using an environmental predicate named emulateConditions. This predicate is initialised to evaluate false. User 'A' now knows they have not yet satisfied the current workflow task and will re-check all sub-tasks are complete.

At heartbeat 6 we change emulateConditions to true, and simulate user 'A' re-requesting activation of the 'Check Data' role. Because this role is being activated within a context (wfc_transformData), two-phase activation is used. The role is confirmed active at line 12.

For the sake of brevity we do not repeat such workflow *conditions* checks. Lines 15 through 22 indicate that user 'A' progresses through the workflow to the 'Check Vendor Account' task.

The diagnostic output from node 'B' at this stage is uninteresting. It has not been the target of any of the commands in table 7.5 except for the broadcast requests to list its active roles.

User 'A' acquires assistance from user 'B'

In this stage of the workflow, we model user 'A' requesting assistance from user 'B' to complete the 'Check Vendor Account' task. User 'B' can assist when delegated active workflow roles from user 'A'. In our testing we have provided user 'A' with privileges that delegate role wf_checkVendorAccount to user 'B'. An alternative approach would be for a single team role manager to delegate and revoke active workflow roles as necessary. Description of the steps we perform at this stage is given in table 7.6.

Turing first to the diagnostic output from the context manager, we see in figure 7.22 lines 21 to 24 that previous role activation confirmation events have indeed set a number of context latches, and filled some contexts such that further role activation requests would trigger negative votes. Line 25 confirms that no roles have been activated at the context manager itself, however.

We now examine the stage two behaviour of node 'A' in figure 7.23. Lines 23 to 25 show

Time	Target	Action
21	A	request delegateCheckVendorAccount privilege
26	A	activate role clientLiaison in context default
31	B	activate role clientLiaison in context default
31	A	revoke credential clientLiaison
31	A	activate role wf_enterOrder in context wfc_checkVendorAccount
36	C	list full contexts
36	C	list latched contexts
36	all	print active roles

Table 7.6: Events in stage two of our workflow scenario

```
-->[21] roleActReq     r=wf_checkVendorAccount s=9001 c=wfc_checkOrderType hb=25
-->[25] roleActConfirm r=wf_checkVendorAccount s=9001 c=wfc_checkOrderType
-->[26] roleActReq     r=clientLiaison s=9000 c=default hb=30
-->[31] roleActReq     r=clientLiaison s=9001 c=default hb=35
-->[31] roleActVote    r=clientLiaison s=9001 c=default w=-1
-->[31] roleActReq     r=wf_enterOrder s=9000 c=wfc_checkVendorAccount hb=35
-->[35] roleActConfirm r=wf_enterOrder s=9000 c=wfc_checkVendorAccount
:::[36] Full contexts: [wfc_transformData, wfc_checkData]
:::[36] Context latches:
        + wfc_checkOrderType:      r=wf_checkVendorAccount
        + wfc_checkVendorAccount:  r=wf_enterOrder
:::[36] Active roles in this database: []
```

Figure 7.22: Diagnostic output from the context manager during stage two

```
23 -->[21] actionReq      p=delegateCheckVendorAccount t=9001
24 <--[21] credForward    r=wf_checkVendorAccount c=wfc_checkOrderType t=9001
25 <--[21] privVote       p=delegateCheckVendorAccount s=9000 w=1
26 -->[26] actionReq      r=clientLiaison s=9000 c=default
27 <--[26] roleActReq     r=clientLiaison s=9000 c=default
28 ---[30] activate:      r=clientLiaison c=default
29 <--[30] roleActConfirm r=clientLiaison s=9000 c=default
30 -->[31] roleActReq     r=clientLiaison s=9001 c=default hb=35
31 <--[31] roleActVote    r=clientLiaison s=9001 c=default w=-1
32 -->[31] credRevoke     r=clientLiaison s=9000 c=default
33 -->[31] actionReq      r=wf_enterOrder s=9000 c=wfc_checkVendorAccount
34 <--[31] roleActReq     r=wf_enterOrder s=9000 c=wfc_checkVendorAccount hb=35
35 ---[35] activate:      r=wf_enterOrder s=9000 c=wfc_checkVendorAccount
36 <--[35] roleActConfirm r=wf_enterOrder s=9000 c=wfc_checkVendorAccount
37 :::[36] Active roles in this database:
38         + role=wf_transformData       c=default
39         + role=wf_checkData           c=wfc_transformData
40         + role=wf_checkOrderType      c=wfc_checkData
41         + role=wf_checkVendorAccount  c=wfc_checkOrderType
42         + role=wf_enterOrder          c=wfc_checkVendorAccount
```

Figure 7.23: Diagnostic output from node 'A' during stage two

the progress of 'A's delegation privilege. The credential forward event causes the actual delegation, and the privilege vote indicates to other interested parties that the request for delegation was successful.

Lines 26 to 29 show 'A's successful two-phase activation of the `clientLiaison` role. This role is not a workflow role, but its activation is predicated on the 'Check Vendor Account' workflow role. The idea here is that liaising with the client is a traditional access control role linked to workflow state.

The `clientLiaison` role exercises a further EDSAC feature – exclusive roles. We are assuming that the process of client liaison here involves contacting the office of the vendor account holder and requesting clarification of some issue. To avoid appearing poorly organised, we ensure that only one principal may be active in the client liaison role and thus in contact with the client at any given time.

To test that this condition is correctly checked, user 'B' is requested also to activate the `clientLiaison` role (figure 7.24 line 9). Lines 30 to 31 show node 'A' rejecting 'B's request to activate this role. We assume that 'A' has received the clarification they need and thus deactivates the liaison role (line 32). They then make workflow role activations to progress to the 'Enter Order' task (36).

The diagnostics shown for node 'B' in figure 7.24 mostly mirror effects from node 'A' already discussed. Lines 4 to 7 show the delegation of 'Check Vendor Account' role. We see 'A's rejection of 'B's request to enter the `clientLiaison` role in line 11. The list of active roles starting on line 12 indicates that, as expected, only the delegated 'Check Vendor Account' role is active at 'B'.

```
-->[21] credForward    r=wf_checkVendorAccount        c=wfc_checkOrderType
<--[21] roleActReq     r=wf_checkVendorAccount s=9001 c=wfc_checkOrderType hb=25
---[25] activate:      r=wf_checkVendorAccount s=9001 c=wfc_checkOrderType
<--[25] roleActConfirm r=wf_checkVendorAccount s=9001 c=wfc_checkOrderType
-->[26] roleActReq     r=clientLiaison         s=9000 c=default            hb=30
-->[31] actionReq      r=clientLiaison         s=9001 c=default
<--[31] roleActReq     r=clientLiaison         s=9001 c=default            hb=35
-->[31] roleActVote    r=clientLiaison         s=9001 c=default w=-1
:::[36] Active roles in this database:
        + role=wf_checkVendorAccount c=wfc_checkOrderType
```

Figure 7.24: Diagnostic output from node ‘B’ during stage two

Time	Target	Action
36	B	activate role `wf_composeRejectMsg` in the `wfc_checkVendorAccount` context
36	B	activate role `wf_enterOrder` in context `wfc_checkVendorAccount`
41	B	activate role `wf_fillOrder` in context `default`
41	A	activate role `wf_composeAcceptance` in context `default`
41	A	activate role `wf_emailConfirmation` in context `default`
41	A	can we enter role `wf_cleanUp` in context `default`?
41	B	request `delegateFillOrder` privilege
41	A	activate role `wf_cleanUp` in context `default`
41	C	list full contexts
41	C	list latched contexts
41	all	print active roles
41	A	request `wf_finish` privilege

Table 7.7: Events in stage three of our workflow scenario

Completing the workflow

This section describes the final tasks of our example workflow. The `AND` split between the ‘Compose Acceptance Message’, ‘Email Confirmation’ and ‘Fill Order Subprocess’ tasks is performed cooperatively between users ‘A’ and ’B’. The commands driving each node in this stage are described in table 7.7.

Towards the end of stage two, node ‘A’ had activated the ‘Enter Order’ workflow role. Note from figure 7.14 that the transition from ‘Check Vendor Account’ to ‘Enter Order’ is within a split of type `XOR`. To demonstrate that the context checks are being correctly performed by the context manager, we request node ‘B’ to activate the `wf_composeRejectMsg` role. This would violate a context latch constraint due to following an invalid workflow task progression. In line 27 of figure 7.25, we see the context manager reject ‘B’s role activation request for this reason. The effect on node ‘B’ is shown in figure 7.27 lines 14 to 16.

```
-->[36] roleActReq     r=wf_composeRejectMsg s=9001 c=wfc_checkVendorAccount hb=40
<--[36] roleActVote    r=wf_composeRejectMsg s=9001 c=wfc_checkVendorAccount w=-1
-->[36] roleActReq     r=wf_enterOrder       s=9001 c=wfc_checkVendorAccount hb=40
-->[40] roleActConfirm r=wf_enterOrder       s=9001 c=wfc_checkVendorAccount
:::[41] Full contexts: [wfc_transformData, wfc_checkData]
:::[41] Context latches:
        + wfc_checkOrderType:      r=wf_checkVendorAccount
        + wfc_checkVendorAccount:  r=wf_enterOrder
:::[41] Active roles in this database: []
```

Figure 7.25: Diagnostic output from the context manager during stage three

We model user ‘A’ completing the ‘Compose Acceptance Message’ (figure 7.26 line 45) and ‘Email Confirmation’ (figure 7.26 line 47) task, but having user ‘B’ complete the ‘Fill Order Subprocess’ (figure 7.27 line 22).

In line 48 of figure 7.26 we see that ‘A’ cannot yet activate the final workflow-role `wf_cleanup` based on the credentials they have active. User ‘B’ has successfully activated the ‘Fill Order Subprocess’ workflow role however, and can request delegation targeted at user ‘A’ via the the `delegateFillOrder` privilege (the credential arrives in line 49).

Given that ‘A’ now has all the prerequisites needed to complete the workflow, it activates the final workflow role and triggers the `wf_finish` privilege, as seen in lines 63 and 64 of figure 7.26. Our workflow instance has been successfully completed.

7.4.4 Obligations and deadlines

In the previous section we demonstrated dynamic constraints restricting the behaviour of sets of principals. Latching context constraints ensured that once one principal moved down a particular workflow path, other principals working on the same workflow instance would have to choose the same path. In fact the mechanisms we have demonstrated can be used to encode obligation policies within EDSAC. Because dynamic constraints were not examined in OASIS research, although time-based conditions can be included in policy rules, OASIS does not support obligation policies.

The nodes activating roles need not be connected with human principals. For instance, one particularly useful computer-controlled role activation function would be to enforce deadlines. Appropriate context constraints can allow policy designers to encode exception cases to ensure timely progression to a workflow conclusion.

We provide an example of a task with a deadline in figure 7.28 drawn from potential policy in a hospital accident and emergency department. When a triage nurse activates the ‘Assessing patient’ role, we also queue a preemptive role activation request for the automatic classification role. We use an `XOR` split from the ‘Assessing patient’ role enforced by the latching context constraints demonstrated in our workflow example. Patients are now guaranteed to be classified in a timely manner. Either the nurse successfully enters a classification, and the automatic classification role activation request fails, or the computer

```
-->[36] roleActVote    r=wf_composeRejectMsg  s=9001 c=wfc_checkVendorAccount w=-1
-->[41] actionReq      r=wf_composeAcceptance s=9000 c=default
<--[41] roleActConfirm r=wf_composeAcceptance s=9000 c=default
-->[41] actionReq      r=wf_emailConfirmation s=9000 c=default
<--[41] roleActConfirm r=wf_emailConfirmation s=9000 c=default
:::[41] cannot reach role wf_cleanup in context default
-->[41] credForward    r=wf_fillOrder              c=default
<--[41] roleActConfirm r=wf_fillOrder         s=9000 c=default
-->[41] actionReq      r=wf_cleanUp           s=9000 c=default
<--[41] roleActConfirm r=wf_cleanUp           s=9000 c=default
:::[41] Active roles in this database:
        + role=wf_transformData       c=default,
        + role=wf_checkData           c=wfc_transformData,
        + role=wf_checkOrderType      c=wfc_checkData,
        + role=wf_checkVendorAccount  c=wfc_checkOrderType,
        + role=wf_enterOrder          c=wfc_checkVendorAccount,
        + role=wf_composeAcceptance   c=default,
        + role=wf_emailConfirmation   c=default,
        + role=wf_fillOrder           c=default,
        + role=wf_cleanUp             c=default,
-->[41] actionReq      p=wf_finish
<--[41] privVote       p=wf_finish            s=9000 w=1
```

Figure 7.26: Diagnostic output from node 'A' during stage three

```
-->[36] actionReq      r=wf_composeRejectMsg s=9001 c=wfc_checkVendorAccount
<--[36] roleActReq     r=wf_composeRejectMsg s=9001 c=wfc_checkVendorAccount hb=40
-->[36] roleActVote    r=wf_composeRejectMsg s=9001 c=wfc_checkVendorAccount w=-1
-->[36] actionReq      r=wf_enterOrder       s=9001 c=wfc_checkVendorAccount
<--[36] roleActReq     r=wf_enterOrder       s=9001 c=wfc_checkVendorAccount hb=40
---[40] activate:      r=wf_enterOrder       s=9001 c=wfc_checkVendorAccount
<--[40] roleActConfirm r=wf_enterOrder       s=9001 c=wfc_checkVendorAccount
-->[41] actionReq      r=wf_fillOrder        s=9001 c=default
<--[41] roleActConfirm r=wf_fillOrder        s=9001 c=default
-->[41] actionReq      p=delegateFillOrder   t=9000
<--[41] credForward    r=wf_fillOrder        t=9000 c=default
<--[41] privVote       p=delegateFillOrder   s=9001 w=1
:::[41] Active roles in this database:
        + role=wf_checkVendorAccount c=wfc_checkOrderType
        + role=wf_enterOrder c=wfc_checkVendorAccount
        + role=wf_fillOrder c=default
```

Figure 7.27: Diagnostic output from node 'B' during stage three

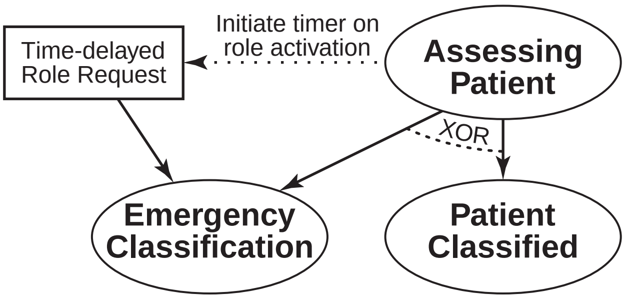


Figure 7.28: Encoding workflow task deadlines

enters the automatic classification role, and the nurse will be unable to enter their own classification. Appropriate roles and privileges can be made available to the nurse from the automatic classification role to allow recovery from this exceptional case and for their work to continue.

Deadlines are only one possible type of obligation policy the EDSAC architecture supports. We can combine deadlines with the cross-principal interactions demonstrated in our workflow example. In this way, we can force principals into an exception case if their actions are not consistent with the actions of other principals in the network, even if the parties do not work together. In a medical example, we might require EHR updates be filed on a central NHS database by hospital staff within a particular duration of time after a doctor has deactivated a role which might have updated these records. If the staff do not do so, they are entered into an exception role that ensures this task is not left neglected.

7.5 Conclusion

This chapter describes our implementation of the EDSAC model, and the technologies we have used to engineer it. Unlike the CBCL OASIS implementation, our focus is on developing small components with loose couplings between them. We have stepped away from the more industry-oriented J2EE application deployment environment to one that reduces our source code to much more accessible units. Even so, we have far from abandoned accepted industry standards. We use an XML policy file format, and make extensive use of XSLT, SVG, and X.509 certificates. Our stream-based communication does not use a message packaging standard like SOAP, but gains the benefit of being much more concise and purpose driven. Prolog itself is an ISO standard, and we avoid using non-ISO constructs wherever possible. Overall, we hope that our development approach will assist future researchers experimenting with and extending our prototype and EDSAC as a whole.

After explaining our choice of Prolog as a development language, and the inter-process communication abstraction we developed for it, we turn to implementation of the EDSAC model itself. We revisit the EDSAC design layers and event types described in chapter 6 discussing the implementation-specific details of our prototype.

We then discuss our research into location-aware access control both in terms of access control to location data, and ongoing experiments into access control on the basis of principal location. We demonstrate our EDSAC prototype making context-aware location disclosure decisions within a simple, distributed access control framework. The content-based nature of EDSAC event delivery ideally suits the flexible deployment structures and distribution of policy rules expected when users take control of the disclosure of their own location.

Truly distributed access control policy requires that networks of access control monitors can interact to maintain a consistent overall state without relying on a centralised conflict management system. We demonstrate our EDSAC implementation's distributed conflict detection and resolution mechanisms by combining distributed access control with workflow management. After examining diagnostic output from our test cases, we discuss how obligations can be encoded into EDSAC policy.

We have now presented and demonstrated most of the key ideas we consider important to create distributed privilege management systems with EDSAC. The next chapter provides overall conclusions and indicates some of the many areas in which future research needs to be directed.

8 Conclusions

This chapter summarises the contributions of this thesis, and discusses the extent to which we have met the requirements for distributed privilege management systems presented in chapter 3. We provide information about some of the ongoing research projects connected with this thesis, and indicate numerous areas that deserve future research attention.

8.1 Contributions

Apart from providing an overview of previous research done into distributed role-based access control (§2), we provide a comprehensive list of features that are important to achieving dynamic access control within large-scale distributed systems (§3). The feature list developed is based on our interactions with other researchers' papers and projects, and our own RBAC deployment experiments.

We have been closely involved with the development of the first successfully deployed OASIS implementation. Combined with our research into policy contexts and distributed policy storage, we feel that the OASIS project has now reached maturity. We are ready to take the accumulated research knowledge from this project and invest it into its successor; the EDSAC project.

A great deal was learnt about the OASIS model in the course of developing its comprehensive implementation. A number of key aspects of the architecture were left (in some cases intentionally) under-specified. Thus, creating the CBCL OASIS implementation involved making specific design decisions relating to how initial authentication should work, and how to transfer credential state between different OASIS processes.

We also learnt many valuable lessons in terms of the development of collaborative projects. The CBCL OASIS NHS EHR prototype was very rapidly completed due to the cohesive interaction of the Opera research group and CBCL (see chapter 5). However ambiguities in the OASIS specification, and changes from the deployment environment originally used in OASIS research, led to adaptation of a number of features that were part of the original OASIS. Some examples: modification for a web environment has meant that the present notion of user sessions is significantly different from that which earlier OASIS publications were based around. XML technology was introduced to manage policy files, and the SOAP intra-OASIS communication. Also a lack of focus on dynamic constraints in the NHS EHR prototype left features such as fast revocation unimplemented.

The RAED project (also discussed in chapter 5) was far less efficient in reaching its conclusion. In this case there were significant delays introduced by problems in the interaction of the various participating research groups. The optimal model for research group cooperation (indicated by the efficiency in developing the EHR prototype) involved a development team that balanced tasks between them and split programming efforts into discrete components. In contrast, in the RAED project the separate researchers and research assistants did not appropriately synchronise their goals and progress reports. As a consequence the significant gaps in understanding slowed down the result. The achievement here was in the re-deployment of CBCL OASIS to the RAED application and working closely with the non-Cambridge project members to ensure that rapid progress was eventually made in the project.

The EHR project was a good exercise in designing scalable applications. The number of potential interacting entities and records being handled was significant. Indeed a similar architecture to our design has been deployed by the NHS in trials that began in August 2004. Observing how OASIS was distributed in practice stimulated many of the design considerations in EDSAC. In particular we focused on providing flexible deployment options, discrete design layers, and using loosely-coupled components with simple, well-defined interfaces between them.

The EHR and RAED projects provided a practical indication of what features are essential in real distributed access control systems. For the EHR project it became clear that the management of digital certificates for authentication was essential, but that the run-time representation of credentials could avoid the need for digital signatures if communicated entirely within the OASIS network. Another key focus was on how OASIS nodes could proxy requests between interacting services, and thus introduce anonymity. The RAED deployment required very few roles, but their parameterisation was crucial. This led to a notable reliance on external databases, and, as a consequence we engineered a much tighter coupling between relational database tables and parameterised role predicates.

A further development came from the difficulties experienced in using the J2EE environment within the RAED project. This led to our seeking of a more accessible development environment for EDSAC. We discuss this implementation in detail in chapter 7.

The focus, and one of the primary contributions of this thesis is the development of the EDSAC architecture (see chapter 6). We have tried to delineate the architecture's components much more clearly than was done for the constituent parts of OASIS, and to emphasise how these components interact with each other, rather than tying down the specification of each of them.

Our prototype, to date, is focused on research environments rather than commercial deployment. However if the EDSAC architecture proves useful in wider distributed access control deployments, this prototype should provide a useful guide to any re-implementation efforts.

8.2 Dynamic privilege management

In this section we review the principles for dynamic privilege management presented in chapter 3. We have made some progress toward many of these principles, and indicate where future research is needed.

Both the OASIS and EDSAC models are clearly distributed in their design and operation (requirement one); they both support the checking of privileges and credentials over a network of access control services. A number of aspects of credential communication in OASIS only became clear in the CBCL OASIS implementation. The EDSAC architecture, on the other hand, explicitly describes the set of events we need to communicate credential state changes. Further, by using a publish/subscribe system for event delivery, we can ensure that undue load is not placed on particular nodes in an access control network.

In terms of supporting dynamic constraints (requirement two), EDSAC far exceeds OASIS and many other distributed RBAC systems. This is in part due to the introduction of the two-phase role activation protocol. By allowing a content-based voting round, distributed nodes can participate in enforcing dynamic constraints without relying on centralised conflict-detection infrastructure. The dynamic constraint capabilities of EDSAC were demonstrated in our workflow management case study (§7.4).

One of the disadvantages to distributed access control schemes based entirely on certificates is that changes to credential validity involve issuing new certificates. In many cases, such as e-commerce server certificates, this is not a problem, as credential validities change infrequently. In applications such as workflow management, however, having to issue and distribute new certificates for extension and revocation is potentially inconvenient. Indeed, if applications do not determine certificate status on-line, they may risk the safety of the system.

OASIS and EDSAC support both certificate-based and light-weight run-time (i.e. session) credentials. The latter are maintained by a dedicated communication infrastructure – the OASIS or EDSAC network. Within EDSAC, by integrating the credential checking function within the event delivery framework of our access control system, we gain significant benefits in terms of the management of groups of credentials. We also avoid the duplication of the event management infrastructure needed to handle dynamic constraints and that handling credential validity, and guarantee both types of events remain synchronised (thus meeting requirement three).

Requirement four involved guaranteeing the termination of policy computation. Doing so requires limiting the potential expressiveness of the policy language employed. While the OASIS inferencing approach will be too limited (or may need unreasonable numbers of environmental predicates) for some deployments, we found it satisfactory for the case studies presented in chapter 5. The policy language chosen for our EDSAC evaluation was closely based on the OASIS language. The additions made to the EDSAC language add the extra latency of the two-phase role activation protocol at most. Apart from this extra latency, our EDSAC prototype provides equivalent termination properties to OASIS.

Policy contexts are our solution for grouping credentials and principals (requirement five). In OASIS contexts were examined for the purpose of policy management and meta-policy considerations. In EDSAC we have used them at run-time too – they are the basis on which we build many of our dynamic constraints. Where sessions collect the credentials relevant for a particular user's interaction with an RBAC system, policy contexts group the interaction and exchange of credentials from other user and computer-controlled sessions. One aspect we definitely need to explore further is how to manage the deployment of policy, however. Whilst this has been discussed in [Bel04], these ideas have not yet been deployed and tested in real access control applications.

Our interaction with the EHR and RAED deployments made it clear that practical access control systems need a means to interact with external software when making policy decisions (requirement six). Any large-scale system will involve decisions being made in which it is impractical to import the entire system-state to the access control system to make it usable without environmental predicates. Taking our CBCL OASIS EHR prototype as an example, there are already thousands of legacy patient record databases in existence. It would be naïve to assume individual policy facets relating to all these records should be stored within the access control system. OASIS always recognised the need for such environmental interactions, and built environmental predicates into its policy language. Indeed parameterised RBAC is most effective only if there is a means to make system-dependent decisions based on role parameter values. In EDSAC we retain a very similar environmental call-out system, although our prototype evaluates Prolog predicates instead of invoking Java methods.

There are two main risks in using environmental predicates. First there is the risk of lack of predicate termination. This can be managed through time-outs in the rule evaluation system. In our EDSAC testing, the external predicates we use are simple enough to guarantee termination. In general a mechanism to ensure automatic failure of environmental predicates that time-out would need to be incorporated within any policy evaluation engine.

The other risk inherent in environmental predicates relates to information flow. We have explored the use of policy contexts to restrict potential information flow when validating parameterised RBAC policy (see §4.4.2 and [BEM03, BEM04]).

One of the unique features of OASIS is its embodiment of an access control middleware, as opposed to a model to engineer into application code. It was always intended that interactions between users, OASIS sessions and OASIS-aware services would occur as a 'control channel' enforcing access to, and the behaviour of the protected resources. The CBCL OASIS implementation couples OASIS with the software specialising EHR sites, but the integration is not static (beyond being Java, OASIS sessions exist as Enterprise JavaBeans). Moreover, the policy files used by CBCL OASIS can be modified and will be reloaded into the OASIS system without requiring any software to be restarted. This loose coupling achieves requirement seven.

Our EDSAC prototype similarly permits dynamic policy redefinition – largely because it is implemented in Prolog. Just as credentials can be asserted and retracted from each Prolog database (in this case operating as the EDSAC session store), so can policy rules. We examine this potential further in section 8.3 below.

In chapter 3 requirement eight related to multi-level policy autonomy – that local policy configuration can be merged with centralised policy rules. Most of the EDSAC discussion in this thesis examines the protocols for dynamic policy interaction, rather than how to ensure that sets of policy rules are managed correctly, thus this principle has not been a focus of our EDSAC work. We discuss this issue further in the Future Work section below (§8.3). We achieved multiple levels of policy autonomy with the RAED project by distinguishing between parameterised rules, and database lookups within them. Thus lower-privilege policy administrators may effect their changes through modifying connected relational databases rather than the role activation rules in the policy files.

Providing a self-administrative policy interface (requirement nine) necessarily ties policy to a particular language. Because we have explicitly avoided doing so in EDSAC, policy self-administration has not been a focus in this implementation. Similar approaches would work with either OASIS or our EDSAC prototype's policy language. Belokosztolszki [Bel04] provides a set of predicates for managing OASIS policy, although we suggest the true administrative requirements for policy maintenance cannot be fully understood without deploying such a system in a real-world application.

Despite this, our EDSAC prototype is shown to manage delegation and revocation of credentials under user control. Because Prolog does not distinguish between such facts and predicates, it would be a simple matter to extend our existing event interface to permit the delegation and revocation of policy rules also.

The last requirement we discussed in chapter 3 was the need for reliable audit information. The CBCL OASIS implementation certainly provides diagnostic output (and lots of it), but the nature of this output has been oriented more towards debugging than audit. As described in section 5.1.3, we added an extra audit interface for the sake of dynamically updating policy visualisations. The challenge, however, is auditing in a distributed environment. The diagnostic output from our EDSAC implementation demonstrates that EDSAC can potentially perform distributed auditing. Our examples have not made explicit use of this, but layer 2 EDSAC design could clearly designate a collection of nodes to monitor events pertaining to certain sets of sessions and contexts. Moreover, the load on these audit nodes can be easily balanced through content-based divisions of the audit subscriptions. Here, publish/subscribe event delivery helps provide efficient aspect-related audit warehouses.

8.3 Future work

This section outlines the numerous avenues for future research highlighted by this thesis. As mentioned in the previous section, we have not yet fully met all the features a distributed access control system should support.

Although we have described the various design layers of EDSAC, and demonstrated the event-based protocol used to communicate policy state, we have not focused on the mechanism by which policy rules are deployed on particular nodes. Distributed policy management involves checking the consistency of the overall integrity in a distributed access control system, as well as managing policy evolution. We have included version attributes for all the elements in our EDSAC prototype, and our Prolog-based system can certainly support run-time modification of policy. However, we have not yet addressed the problems involved in making a transition from one version to another while the access control system continues to operate. In such circumstances credentials of different versions may be active simultaneously.

The two-phase role activation protocol involves nodes avoiding planned activations if they have received any negative role activation votes. However, the notion of voting on role activation has far more widespread potential. We can easily extend EDSAC to support policy that explicitly specifies the minimum number of positive votes, and where they must come from, in order to activate a role (as shown in the augmented rules presented in section 4.3.3). We foresee the utility of protocols that can incorporate both positive and negative votes to determine whether to proceed or not with role activation. Proceeding in the face of negative votes may involve further actions on the part of those who cast the negative votes. Protocols that accumulate votes will need high-level design principles to guarantee the stability of the policy network e.g. precluding two nodes constantly flipping between a mutually exclusive role activation.

We have touched on the idea of multi-phase negotiation but have not attempted to incorporate dynamic conflict resolution in our EDSAC prototype. It would be possible to deploy policy that contains intermediate roles used as steps in a multiple-round role activation negotiation. Our event protocols do not explicitly manage how a node might find the non-local basis for a particular role activation rejection. Such information is likely to be useful for driving user interfaces in domains where the principals are not policy experts. Further, in applications such as trust-based access control within the SECURE project (continuing from [DBE$^+$04]), the activating node may be able to negotiate a compromise with the rejecting node.

Real-world EDSAC deployments are likely to lead to significant advances beyond our prototype in terms of further development of policy design and monitoring tools. The ongoing LCE location-aware access control project is expected to develop some forms of application-specific visualisation (e.g. choosing ‘dark zones’ using a clickable office map), but we would also like to experiment further with visualisation techniques for the policy definition itself.

We are also keen to test the integration of different role-aware policy languages within the EDSAC architecture. In particular, we plan to incorporate the Cassandra language [BS04b, BS04a]. Apart from providing expressive but predictable execution, its designers are close to hand within the University of Cambridge Computer Laboratory.

There are a number of ongoing projects in the early stages of development at the time of writing. Some of these projects intend to use features available in the OASIS system, but the EDSAC model will fit their OASIS needs without modification, and provide extra capabilities. Projects we have not already mentioned include the use of OASIS technology within the Integration Broker for Heterogeneous Information Sources (IBHIS); and federating security between different systems within the Police Information Technology Organisation (PITO). The IBHIS project is a collaboration between the University of Manchester Institute of Science and Technology, Keele University and the University of

Durham. It aims to use a 'software as a service' methodology to support information integration within the health care domain.

8.4 Conclusion

This thesis has explored dynamic privilege management in distributed access control systems. We describe deployments of the first stable OASIS implementation, and further research performed in consequence. Based on our experiences with OASIS we developed the Event-driven Distributed Scalable Authorisation Control model. Beyond providing all the functionality of OASIS policy, EDSAC can enforce distributed dynamic RBAC constraints and obligations.

By focusing on the interaction between components at four distinct design layers, we allow great flexibility in the way particular event delivery mechanisms, inference engines and authentication mechanisms can be integrated. We demonstrate EDSAC with a series of case studies and indicate the areas in which we intend to continue our distributed access control research.

Bibliography

[AB00] Alan S. Abrahams and J. Bacon. Event-centric business rules in e-commerce applications. In *Workshop on Best Practices in Business Rule Design and Implementation at the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Minneapolis, MN, October 2000.

[AB01a] Alan S. Abrahams and Jean M. Bacon. Event-centric policy specification for e-commerce applications. In *Workshop on Policies for Distributed Systems and Networks*, Bristol, UK, January 2001.

[AB01b] Alan S. Abrahams and Jean M. Bacon. Occurrence-centric policy specification for e-commerce applications. In *Workshop on Formal Modelling for Electronic Commerce*, Norway, Oslo, June 2001.

[AB01c] Alan S. Abrahams and Jean M. Bacon. Representing and enforcing e-commerce contracts using occurrences. In *Proceedings of the Fourth International Conference on Electronic Commerce Research, Edwin L. Cox School of Business, Southern Methodist University, Dallas, Texas, USA*, November 2001.

[AB01d] Alan S. Abrahams and Jean M. Bacon. Representing and enforcing electronic commerce contracts over a wide range of platforms using occurrence stores. In *Fourth CaberNet Plenary Workshop*, Pisa, Italy, October 2001.

[AB02a] Alan S. Abrahams and Jean M. Bacon. The life and times of identified, situated, and conflicting norms. In *Procedings of the Sixth International Workshop on Deontic Logic in Computer Science (DEON'02), Imperial College, London, UK*, May 2002.

[AB02b] Alan S. Abrahams and Jean M. Bacon. A software implementation of Kimbrough's disquotation theory for representing and enforcing electronic commerce contracts. *Group Decision and Negotiations Journal*, 11(6):1–38, November 2002.

[Abr02] Alan S. Abrahams. *Developing and Executing Electronic Commerce Applications with Occurrences.* PhD thesis, University of Cambridge Computer Laboratory, 2002.

[ACM01] Vijayalakshmi Atluri, Soon A. Chun, and Pietro Mazzoleni. A Chinese Wall security model for decentralized workflow systems. In *Proceedings of the Eighth ACM conference on Computer and Communications Security*, pages 48–57. ACM Press, 2001.

[AEB02a] Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. An asynchronous rule-based approach for business process automation using obligations. In *Proceedings of the Third ACM SIGPLAN Workshop on Rule-Based Programming (RULE'02)*, Pittsburgh, USA, October 2002.

[AEB02b] Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. A coverage-determination mechanism for checking business contracts against organizational policies. In *Proceedings of the Third VLDB Workshop on Technologies for E-Services (TES'02)*, 2002.

[AEB02c] Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. Mechanical consistency analysis for business contracts and policies. In *Proceedings of the Fifth International Conference on Electronic Commerce Research*, Montreal, Canada, October 2002.

[AEB04a] Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. An event-based paradigm for e-commerce application specification and execution. In *Proceedings of the Seventh International Conference on Electronic Commerce Research (ICECR7)*, Dallas, Texas, June 2004.

[AEB04b] Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. Practical contract storage, checking, and enforcement for business process automation. In Steven O. Kimbrough and D.J. Wu, editors, *Formal Modeling for Electronic Commerce: Representation, Inference, and Strategic Interaction*, pages 33–77. Springer-Verlag, 2004.

[AEB04c] Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. Towards a benchmark for e-contract checking and monitoring. In *Proceedings of the Seventh International Conference on Electronic Commerce Research (ICECR7)*, Dallas, Texas, June 2004.

[AK91] Hassan Aït-Kaci. *Warren's Abstract Machine, A Tutorial Reconstruction.* MIT Press, 1991.

[AK02] Alan S. Abrahams and Steven O. Kimbrough. Treating disjunctive obligation and conjunctive action in event semantics with disquotation. *Wharton Business School Working Paper Series*, 2002.

[AM04] Xuhui Ao and Naftaly H. Minsky. On the role of roles: from role-based to role-sensitive access control. In *Proceedings of the Ninth ACM Symposium on Access Control Models And Technologies*, pages 51–60. ACM Press, 2004.

[And01] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, April 2001.

[And04] Anne Anderson. An introduction to the web services policy language (WSPL). In *Policy 2004: IEEE Fifth International Workshop on Policies for Distributed Systems and Networks*, 2004.

[AS03] Mohammad Bin Abdullah and Ravi S. Sandhu. Induced role hierarchies with attributes-based RBAC. In *Procedings of the Eighth ACM Symposium on Access Control Models and Technologies*, 2003.

[ASFa] Apache Software Foundation. The Apache XML Project. `http://httpd.apache.org/ABOUT_APACHE.html`.

[ASFb] Apache Software Foundation. The Apache XML Project. `http://xml.apache.org/`.

[ASFc] Apache Software Foundation. The Batik SVG Toolkit. `http://xml.apache.org/batik/`.

[BB03] Vijay G. Bharadwaj and John S. Baras. Towards automated negotiation of access control policies. In *Policy 2003: IEEE Fourth International Workshop on Policies for Distributed Systems and Networks*, 2003.

[BBB97] Dixie B. Baker, Robert M. Barnhart, and Teresa T. Buss. PCASSO: applying and extending state-of-the-art security in the healthcare domain. In *Proceedings of the Thirteenth Annual Computer Security Applications Conference*, page 251. IEEE Computer Society, 1997.

[BBF01] Elisa Bertino, Piero A. Bonatti, and Elena Ferrari. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security*, 4(3):191–223, August 2001.

[BBHM95] Jean M. Bacon, John Bates, Richard Hayton, and Ken Moody. Using events to build distributed applications. In *Proceedings of the IEEE Services in Distributed and Networked Environments Workshop (SDNE'95)*, pages 149–155, June 1995.

[BE03] András Belokosztolszki and David M. Eyers. Shielding RBAC infrastructures from cyberterrorism. In *Research Directions in Data and Applications Security*, pages 3–14. Kluwer Academic Publishers, 2003.

[Bek03] Scott Bekker. Microsoft: No security patches in December. `http://www.entmag.com/news/article.asp?EditorialsID=6060`, December 2003.

[Bel04] András Belokosztolszki. Role-Based Access Control policy administration. Technical Report UCAM-CL-TR-586, University of Cambridge, Computer Laboratory, March 2004.

[BEM03] András Belokosztolszki, David M. Eyers, and Ken Moody. Policy contexts: Controlling information flow in parameterised RBAC. In *Policy 2003: IEEE Fourth International Workshop on Policies for Distributed Systems and Networks*, 2003.

[BEM04] András Belokosztolszki, David M. Eyers, and Ken Moody. A formal model for hierarchical policy contexts. In *Policy 2004: IEEE Fifth International Workshop on Policies for Distributed Systems and Networks*, pages 127–136, June 2004.

[BEP$^+$03] András Belokosztolszki, David M. Eyers, Peter R. Pietzuch, Jean Bacon, and Ken Moody. Role-based access control for publish/subscribe middleware architectures. In *International Workshop on Distributed Event-Based Systems (DEBS03)*, 2003.

[BEWM03] András Belokosztolszki, David M. Eyers, Wei Wang, and Ken Moody. Policy storage for role-based access control systems. In *Proceedings of the Twelvth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03)*, 2003.

[BFA97] Elisa Bertino, Elena Ferrari, and Vijayalakshmi Atluri. A flexible model supporting the specification and enforcement of role-based authorization in workflow management systems. In *Proceedings of the second ACM workshop on Role-Based Access Control*, pages 1–12. ACM Press, 1997.

[BFK99] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote: Trust management for public-key infrastructures. In Bruce Christianson, Bruno Crispo, William S. Harbison, and Michael Roe, editors, *Security Protocols*, number 1550 in Lecture Notes in Computer Science, pages 59–66, Cambridge, United Kingdom, April 1999.

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1996.

[BFS03] Elisa Bertino, Elena Ferrari, and Anna C. Squicciarini. Trust-X: an XML framework for trust negotiations. In *Policy 2003: IEEE Fourth International Workshop on Policies for Distributed Systems and Networks*, 2003.

[BGL$^+$03] Keith H. Bennett, Nicolas E. Gold, Paul J. Layzell, Fujun Zhu, Pearl Brereton, David Budgen, John Keane, Ioannis A. Kotsiopoulos, Mark Turner, Jie

Xu, Orouba Almilaji, Jung-Ching Chen, and Ali Owrak. A broker architecture for integrating data using a web services environment. In *Proceedings of the First International Conference on Service-Oriented Computing*, Trento, Italy, December 2003.

[Bha03] Rafae Bhatti. X-GTRBAC: An XML-based policy specification framework and architecture for enterprise-wide access control. Technical Report 2003-27, CERIAS, Purdue University, May 2003.

[BI98] Christophe Bidan and Valérie Issarny. Dealing with multi-policy security in large open distributed systems. *Lecture Notes in Computer Science*, 1485, 1998.

[Bib77] Ken J. Biba. Integrity considerations for secure computer systems. Technical Report TR-3153, Mitre, Bedford, MA, April 1977.

[BJBG04] Rafae Bhatti, James Joshi, Elisa Bertino, and Arif Ghafoor. X-GTRBAC admin: a decentralized administration model for enterprise wide access control. In *Proceedings of the Ninth ACM Symposium on Access Control Models And Technologies*, pages 78–86. ACM Press, 2004.

[BKR$^+$98] Ulrik Brandes, Patrick Kenis, Jörg Raab, Volker Schneider, and Dorothea Wagner. Explorations into the visualization of policy networks, 1998.

[BL73] David E. Bell and Leonard J. La Padula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, The MITRE Corp., Bedford MA, May 1973.

[BL98] Tim Berners-Lee. A roadmap to the Semantic Web. `http://www.w3.org/DesignIssues/Semantic.html`, September 1998.

[Bla98] Blackboard Inc. Blackboard. `http://www.blackboard.com/`, July 1998.

[BLM01] Jean M. Bacon, Michael Lloyd, and Ken Moody. Translating role-based access control policy within context. *Lecture Notes in Computer Science*, 1995, 2001.

[BM02a] András Belokosztolszki and Ken Moody. Meta-policies for distributed role-based access control systems. In *Third International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, June 2002.

[BM02b] András Belokosztolszki and Ken Moody. Meta-policies for distributed role-based access control systems. In *Policy 2002: IEEE Third International Workshop on Policies for Distributed Systems and Networks*, pages 106–115, 2002.

[BMB$^+$00] Jean M. Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel, and Mark Spiteri. Generic support for distributed applications. *IEEE Computer*, 33(3):68–76, March 2000.

[BMCO03] Jean M. Bacon, Ken Moody, David W. Chadwick, and Alexander Otenko. Persistent versus dynamic role membership. In *Seventeenth IFIP WG3 Annual Working Conference on Data and Application Security*. IFIP, August 2003.

[BMY01] Jean M. Bacon, Ken Moody, and Walt Yao. Access control and trust in the use of widely distributed services. In *Middleware 2001*, volume 2218, pages 300–315, November 2001.

[BMY02a] Jean Bacon, Ken Moody, and Walt Yao. A model of OASIS Role-Based Access Control and its support for active security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, November 2002.

[BMY02b] Jean M. Bacon, Ken Moody, and Walt Yao. A model of OASIS Role-Based Access Control and its support for active security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, November 2002.

[BN89] David F. C. Brewer and Michael J. Nash. The Chinese Wall security policy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[BS03] Alastair R. Beresford and Frank Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2(1):46–55, 2003.

[BS04a] Moritz Y. Becker and Peter Sewell. Cassandra: distributed access control policies with tunable expressiveness. In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, June 2004.

[BS04b] Moritz Y. Becker and Peter Sewell. Cassandra: flexible trust management, applied to electronic health records. In *Proceedings of the Seventeenth IEEE Computer Security Foundations Workshop*, June 2004.

[Car95] Mats Carlsson. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, 1995.

[Car98] Antonio Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, Milano, Italy, December 1998.

[CB02] Richard Clayton and Mike Bond. Experience using a low-cost FPGA design to crack DES keys. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, August 2002.

[CH01] Daniel Cabeza and Manuel Hermenegildo. Distributed WWW programming using Prolog and the PiLLoW library. *Theory and Practice of Logic Programming*, 1(3):251–282, May 2001.

[Cha03] David W. Chadwick. Deficiencies in LDAP when used to support PKI. *Communications of the ACM*, 46(3):99–104, 2003.

[Cho04] Shih-Chien Chou. LnRBAC: A multiple-levelled role-based access control model for protecting privacy in object-oriented systems. *Journal of Object Technology*, 3(3):91–120, March-April 2004.

[CHV00] Daniel Cabeza, Manuel Hermenegildo, and Sacha Varma. The Ciao Prolog system - HTML/XML/CGI programming. `http://www.clip.dia.fi.upm.es/Software/Ciao/ciao_html/ciao_110.html`, July 2000.

[CLS$^+$01] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dev, Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles. In *Procedings of the Sixth ACM Symposium on Access Control Models and Technologies*, pages 10–20, 2001.

[CM94] William F. Clocksin and Christopher S. Mellish. *Programming in Prolog.* Springer-Verlag, fourth edition, 1994.

[CM03] David W. Chadwick and Darren Mundy. Policy based electronic transmission of prescriptions. In *Policy 2003: IEEE Fourth International Workshop on Policies for Distributed Systems and Networks*, 2003.

[CN01] Gianpaolo Cugola and Elisabetta Di Nitto. The JEDI event-based infrastructure and its applications to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering (TSE)*, 27(9):825–850, September 2001.

[CO02a] David W. Chadwick and Alexander Otenko. The PERMIS X.509 role based privilege management infrastructure. In *Proceedings of the Seventh ACM Symposium on Access Control Models And Technologies*, pages 135–140. ACM Press, 2002.

[CO02b] David W. Chadwick and Alexander Otenko. RBAC policies in XML for X.509 based privilege management. In *Procedings of the Seventeenth International Conference on Information Security*, 2002.

[Cou04] George Coulouris. Exploring the design space: A brief overview of ubiquitous computing research at the laboratory for communication engineering, cambridge. In *Proceedings of the Ubiquitous Computing Conference (Ubiconf)*, Gresham College, London, 2004.

[Cra03] Jason Crampton. Specifying and enforcing constraints in role-based access control. In *Procedings of the Eighth ACM Symposium on Access Control Models and Technologies*, 2003.

[CRW99] Antonio Carzaniga, David R. Rosenblum, and Alexander L. Wolf. Challenges for distributed event services: Scalability vs. expressiveness. In Wolfgang Emmerich and Volker Gruhn, editors, *ICSE '99 Workshop on Engineering Distributed Objects (EDO '99)*, May 1999.

[CRW01] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[CTWS02] Eve Cohen, Roshan Thomas, William Winsborough, and Deborah Shands. Models for Coalition-Based Access Control (CBAC). In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies (SACMAT-02)*, pages 97–106, New York, June 3–4 2002.

[CW87] David C. Clark and David R. Wilson. A comparison of commercial and military security policies. In *Proceedings of the IEEE Symposium on Security and Privacy, Washington DC*, 1987.

[Dam02] Nicodemos C. Damianou. *A Policy Framework for Management of Distributed Systems*. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, February 2002.

[DBC+00] David Durham, Jim Boyle, Ron Cohen, Shai Herzog, Raju Rajan, and Arun Sastry. RFC 2748: The COPS (common open policy service) protocol, January 2000.

[DBE+04] Nathan Dimmock, András Belokosztolszki, David M. Eyers, Jean M. Bacon, and Ken Moody. Using trust and risk in role-based access control policies. In *Proceedings of the Ninth ACM Symposium on Access Control Models And Technologies*, pages 156–162. ACM Press, 2004.

[DDLS01] Nicodemos C. Damianou, Naranker Dulay, Emil C. Lupu, and Morris S. Sloman. The Ponder policy specification language. *Lecture Notes in Computer Science*, 1995:18–38, 2001.

[DDLS02] Naranker Dulay, Nicodemos C. Damianou, Emil C. Lupu, and Morris S. Sloman. A policy language for the management of distributed agents. *Lecture Notes in Computer Science*, 2222, 2002.

[Den76] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[DH98] Steve Deering and Robert Hinden. RFC 2460: Internet Protocol, Version 6 (IPv6) specification, December 1998.

[Die99] Tim Dierks. The TLS protocol version 1.0. `http://www.ietf.org/rfc/rfc2246.txt`, January 1999.

[Dim03] Nathan Dimmock. How much is ‘enough’? Risk in trust-based access control. In *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises: Enterprise Security (Special Session on Trust Management)*, pages 281–282, June 2003.

[EFGK03] Patrick T. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[EFL$^+$99] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certicate theory. `http://www.ietf.org/rfc/rfc2693.txt`, 1999.

[EGK$^+$02] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz - open source graph drawing tools. In *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 483–484. Springer, 2002.

[EM03] David M. Eyers and Ken Moody. Credential negotiation with limited disclosure via iterative range refinement in an unordered space. In *Proceedings of the Fourteenth International Workshop on Database and Expert Systems Applications*, page 427. IEEE Computer Society, 2003.

[ESW02] David M. Eyers, John A. Shepherd, and Raymond Wong. Merging Prolog and XML databases. In *Proceedings of the Seventh Australasian Document Computing Symposium (ADCS’02)*, December 2002.

[FK92] David Ferraiolo and Richard Kuhn. Role-based access controls. In *Fifteenth NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[FPP$^+$01] Eric Freudenthal, Tracy Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti. dRBAC: Distributed role-based access control for dynamic coalition environments. Technical Report TR2001-819, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, 2001.

[FPP$^+$02] Eric Freudenthal, Tracy Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti. dRBAC: Distributed role-based access control for dynamic coalition environments. In *Proceedings of the Twenty-Second IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 411–420, Vienna, Austria, 2002.

[GBG$^+$96] William Grimson, Damon Berry, Jane Grimson, Gaye Stephens, Eoghan Felton, Peter Given, and Rory O'Moore. Technical description: Federated healthcare record server - the Synapses paradigm. `http://www.cs.tcd.ie/synapses/public/html/technicaldescription.html`, 1996.

[GI96] Luigi Giuri and Pietro Iglio. A formal model for role-based access control with constraints. In *Proceedings of the Ninth IEEE Computer Security Foundations Workshop*, page 136. IEEE Computer Society, 1996.

[GN00] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, 30(11):1203–1233, 2000.

[Gol99] Dieter Gollmann. *Computer Security.* John Wiley & Sons, 1999.

[Hay96] Richard Hayton. *OASIS An Open Architecture for Secure Interworking Services.* PhD thesis, University of Cambridge, 1996.

[HBS$^+$02] Mark Hapner, Rich Burridge, Rahul Sharma, Joseph Fialli, and Kate Stout. Java Message Service. `http://java.sun.com/products/jms/docs.html`, April 2002.

[HFPS99] Russell Housley, Warwick Ford, Tim Polk, and David Solo. RFC 2459: Internet X.509 public key infrastructure certificate and CRL profile, January 1999.

[HM04] John Hughes and Eve Maler. Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1.1. `http://www.oasis-open.org/committees/download.php/6837/sstc-saml-tech-overview-1.1-cd.pdf`, May 2004.

[Hom02] Alexis Hombrecher. *Reconciling Event Taxonomies Across Administrative Domains.* PhD thesis, University of Cambridge Computer Laboratory, 2002.

[HRU76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. In *Communications of the ACM*, pages 461–471. ACM, August 1976.

[HS04] Urs Hengartner and Peter Steenkiste. Implementing access control to people location information. In *Proceedings of the Ninth ACM Symposium on Access Control Models And Technologies*, pages 11–20. ACM Press, 2004.

[HYBM00] John H. Hine, Walt Yao, Jean M. Bacon, and Ken Moody. An architecture for distributed OASIS services. In *Middleware*, pages 104–120, 2000.

[IBM01] IBM TJ Watson Research Center. Gryphon: Publish/Subscribe over Public Networks. `http://researchweb.watson.ibm.com/gryphon/Gryphon`, December 2001.

[IBM02] IBM Corporation. WebSphere MQ Event Broker. `http://www.ibm.com/software/integration/mqfamily/eventbroker/`, May 2002.

[IF01] Ironflare AB. The Orion Application Server. `http://www.orionserver.com/`, 2001.

[ISO95] International Organization for Standardization ISO. *ISO/IEC 13211-1:1995: Information technology — Programming languages — Prolog — Part 1: General core.* International Organization for Standardization, Geneva, Switzerland, 1995.

[JBG02] James B. D. Joshi, Elisa Bertino, and Arif Ghafoor. Temporal hierarchies and inheritance semantics for GTRBAC. In *Proceedings of the Seventh ACM Symposium on Access Control Models And Technologies*, pages 74–83. ACM Press, 2002.

[JBLG01] James B. D. Joshi, Elisa Bertino, Usman Latif, and Arif Ghafoor. Generalized temporal role based access control model (GTRBAC) part 1: specification and modeling. Technical Report 2001-47, CERIAS, Purdue University, 2001.

[JBSG03] James B. D. Joshi, Elisa Bertino, Basit Sahfiq, and Arif Ghafoor. Dependencies and separation of duty constraints in GTRBAC. In *Procedings of the Eighth ACM Symposium on Access Control Models and Technologies*, 2003.

[JEZ02] Trent Jaeger, Antony Edwards, and Xiaolan Zhang. Managing access control policies using access control spaces. In *Proceedings of the Seventh ACM Symposium on Access Control Models And Technologies*, pages 3–12. ACM Press, 2002.

[Kag02] Lalana Kagal. Rei: A policy language for the me-centric project. Technical Report HPL-2002-270, Hewlett Packard Laboratories, October 04 2002.

[KCK01] Eleanna Kafeza, Dickson K.W. Chiu, and Irene Kafeza. View-based contracts in an e-service cross-organizational workflow environment. In *Proceedings of the Second International Workshop on Technologies for E-Services (TES'01)*, pages 74–88. Springer, 2001.

[KGV99] Marjanca Koetsier, Paul W. P. J. Grefen, and Jochem Vonk. Cross-organisational workflow: CrossFlow ESPRIT E/28635 contract model, deliverable D4b. Technical report, CrossFlow consortium, 1999.

[KKSM02] Axel Kern, Martin Kuhlmann, Andreas Schaad, and Jonathan D. Moffett. Observations on the role life-cycle in the context of enterprise security management. In *Proceedings of the Seventh ACM Symposium on Access Control Models And Technologies*, pages 43–51. ACM Press, 2002.

[Kno00] Konstantin Knorr. Dynamic access control through Petri Net workflows. In *Proceedings of the Sixteenth Annual Computer Security Applications Conference (ACSAC)*, pages 159–167, New Orleans, LA, December 2000.

[KRSR98] Gerti Kappel, Stefan Rausch-Schott, and Werner Retschitzegger. Coordination in workflow management systems – a rule-based approach. In Wolfram Conen and Gustaf Neumann, editors, *Coordination Technology for Collaborative Applications – Organizations, Processes, and Agents*, pages 99–120. Springer LNCS 1364, 1998.

[KS99] Savith Kandala and Ravi S. Sandhu. Extending the BFA workflow authorization model to express weighted voting. In *IFIP Workshop on Database Security*, pages 145–159, 1999.

[KSS03] Martin Kuhlmann, Gerhard Schimpf, and Dalia Shohat. Role mining - revealing business roles for security administration using data mining technology. In *Procedings of the Eighth ACM Symposium on Access Control Models and Technologies*, 2003.

[Kuh97] Richard Kuhn. Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. In *Proceedings of the Second ACM Workshop on Role-Based Access Control (RBAC-97)*, pages 23–30, New York, November 6–7 1997. ACM Press.

[Lam71] Butler Lampson. Protection. In *Proceedings of the Fifth Annual Princeton Conference on Information Sciences and Systems*, pages 437–443, Princeton University, 1971.

[Lev84] Henry M. Levy. *Capability-based Computer Systems.* Digital Press, 1984.

[LLCT00] Jim J. Longstaff, Mike A. Lockyer, Graham Capper, and Michael G. Thick. A model of accountability, confidentiality and override for healthcare and other applications. In *Proceedings of the Fifth ACM Workshop on Role-Based Access Control (RBAC'00)*, pages 71–76. ACM Press, July 2000.

[LLN03a] Jim J. Longstaff, Mike A. Lockyer, and John Nicholas. Authorisation models for complex computing applications. In *Proceedings of the Information Security Solutions Europe conference*, 2003.

[LLN03b] Jim J. Longstaff, Mike A. Lockyer, and John Nicholas. The Tees Confidentiality Model: an authorization model for identities and roles. In *Procedings* *of the Eighth ACM Symposium on Access Control Models and Technologies*, 2003.

[LS99] Emil C. Lupu and Morris S. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, November/December 1999.

[LW88] Frederick H. Lochowsky and Carson C. Woo. Role-based security in database management systems. In Carl E. Landwehr, editor, *Database Security: Status and Prospects*, pages 209–222, Amsterdam, The Netherlands, 1988. North-Holland Publishing Co.

[LWS$^+$04] Dawn Leeder, Heather Wharrad, Nicki Stephens, Veena Rodrigues, John Mclachlan, and Patrick Mcelduff. Universities' collaboration in e-learning (UCeL): a virtual community of practice in health professional education. In *In Proceedings of the IADIS International Conference on Web Based Communities (WBC2004)*, Lisbon, Portugal, March 2004.

[MBBC02] Daniel Masys, Dixie B. Baker, Amy Butros, and Kevin E. Cowles. Giving patients access to their medical records via the internet: The PCASSO experience. *American Medical Informatics Association*, 9(2):181–191, 2002.

[McD03] Patrick McDaniel. On context in authorization policy. In *Procedings of the Eighth ACM Symposium on Access Control Models and Technologies*, 2003.

[McL85] John McLean. A comment on the 'basic security theorem' of Bell and LaPadula. *Information Processing Letters*, 20(2):67–70, February 1985.

[McL90] John McLean. The specification and modeling of computer security. *Computer*, 23(1):9–16, January 1990.

[MESW01] Bob Moore, Ed Ellesson, John Strassner, and Andrea Westerinen. RFC 3060: Policy core information model – version 1 specification, February 2001.

[MFD03] Ginger Myles, Adrian Friday, and Nigel Davies. Preserving privacy in environments with location-based applications. *IEEE Pervasive Computing*, 2(1):56–64, 2003.

[Mic02] Microsoft. Defining the basic elements of .NET. `http://www.microsoft.com/net/basics/whatis.asp`, April 2002.

[Mik02] Zoltán Miklós. Towards an access control mechanism for wide-area publish/subscribe systems. In *International Workshop on Distributed Event-based Systems*, July 2002.

[Min91] Naftaly H. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, 17(2):183–195, 1991.

[MN99] Terry J. Mayes and Irene Neilson. Learning from other people's dialogues: questions about computer-based answers. *Innovative Adult Learning with Innovative Technologies*, 1:31–47, 1999.

[MS93] Jonathan D. Moffett and Morris S. Sloman. Policy conflict analysis in distributed system management. *Journal of Organizational Computing*, April 1993.

[MU00] Naftaly H. Minsky and Victoria Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology*, 9(3):273–305, 2000.

[NC00] SangYeob Na and SuhHyun Cheon. Role delegation in role-based access control. In *Proceedings of the Fifth ACM workshop on Role-Based Access Control*, pages 39–44, 2000.

[NHS98] Department of Health, NHS Executive. Information for health: An information strategy for the modern NHS 1998-2005. `http://www.dh.gov.uk/assetRoot/04/01/43/89/04014389.pdf`, January 1998.

[NLE$^+$04] Steve Neely, Helen Lowe, David M. Eyers, Jean M. Bacon, Julian Newman, and Xiaofeng Gong. An architecture for supporting vicarious learning in a distributed environment. In *Proceedings of the 2004 ACM symposium on Applied Computing*, pages 963–970. ACM Press, 2004.

[NLN$^+$04] Julian Newman, Helen Lowe, Steve Neely, Xiaofeng Gong, David M. Eyers, and Jean M. Bacon. A tutorial task and tertiary courseware model for collaborative learning communities. *Electronic Journal on E-Learning*, 2(1):159–166, March 2004.

[NO99] Matunda Nyanchama and Sylvia L. Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):3–33, 1999.

[NS02] Gustaf Neumann and Mark Strembeck. A scenario-driven role engineering process for functional RBAC roles. In *Proceedings of the Seventh ACM Symposium on Access Control Models And Technologies*, pages 33–42. ACM Press, 2002.

[OAS02] Organization for the Advancement of Structured Information Standards (OASIS) Extensible Access Control Markup Language Technical Committee. Collaboration-Protocol Profile and Agreement Specification. `http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf`, 2002.

[OAS03] Organization for the Advancement of Structured Information Standards (OASIS). OASIS Open Technical Committee Process. `http://www.oasis-open.org/committees/process.php`, August 2003.

[OMG02] Object Management Group. The Common Object Request Broker Architecture: Core Specification, Revision 3.0, December 2002.

[Osb02] Sylvia L. Osborn. Information flow analysis of an RBAC system. In *Seventh ACM Symposium on Access Control Models and Technologies*, pages 163–168. ACM Press, 2002.

[PB02] Peter R. Pietzuch and Jean M. Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of the First International Workshop on Distributed Event-Based Systems (DEBS'02)*, Vienna, Austria, July 2002.

[PD94] Norman W. Paton and Oscar Diaz. Active database systems. *ACM Computing Surveys*, 1(31), 1994.

[PHB02] Tim Polk, Russell Housley, and Larry Bassham. RFC 3279: Algorithms and identifiers for the internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile, April 2002.

[Pie04] Peter R. Pietzuch. Hermes: A scalable event-based middleware. Technical Report UCAM-CL-TR-590, University of Cambridge, Computer Laboratory, June 2004.

[Pos81] Jon Postel. RFC 791: Internet Protocol, September 1981.

[Pow96] David Powell. Group communication. *Communications of the ACM*, 39(4):50–53, 1996.

[PS02] Jaehong Park and Ravi S. Sandhu. Towards usage control models: beyond traditional access control. In *Proceedings of the Seventh ACM Symposium on Access Control Models And Technologies*, pages 57–64. ACM Press, 2002.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[RSFWH98] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[RWRS00] Carl Rigney, Steve Willens, Alan Rubens, and William Simpson. RFC 2865: Remote authentication dial in user service (RADIUS), June 2000.

[SA03] Dongwas Shin and Gail-Joon Ahn. On modeling system centric information for role engineering. In *Procedings of the Eighth ACM Symposium on Access Control Models and Technologies*, 2003.

[San88] Ravi S. Sandhu. Transaction control expressions for separation of duties. In *In Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pages 282–286, December 1988.

[San95] Ravi S. Sandhu. Roles versus groups. In *Proceedings of the First ACM workshop on Role-Based Access Control*, pages I–25–26, 1995.

[San96] Ravi S. Sandhu. Access control: The neglected frontier. In *Australasian Conference on Information Security and Privacy*, pages 219–227, 1996.

[SBM99] Ravi S. Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and System Security*, 2(1):105–135, February 1999.

[SBM03] David Scott, Alastair Beresford, and Alan Mycroft. Spatial security policies for mobile agents in a sentient computing environment. In *Fundamental Approaches to Software Engineering (FASE) 2003*, volume 2621 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2003.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[Sch99] Marc-Thomas Schmidt. The evolution of workflow standards. *IEEE Concurrency*, 1999.

[SFK00] Ravi S. Sandhu, David F. Ferraiolo, and Richard Kuhn. The NIST model for role-based access control: towards a unified standard. In *Proceedings of the Fifth ACM workshop on Role-Based Access Control*, pages 47–63, 2000.

[Sha01] Bill Shannon. Java 2 platform enterprise edition specification, v1.3. `http://java.sun.com/j2ee/j2ee-1_3-fr-spec.pdf`, 2001.

[SHC96] Zoltan Somogyi, Fergus Henderson, and Thomas Conway. The execution algorithm of Mercury, an efficient purely declarative logic programming language. *Journal of Logic Programming*, 29(1-3):17–64, 1996.

[SHP89] Michael Stonebraker, Marti A. Hearst, and Spyros Potamianos. A commentary on the POSTGRES rule system. *SIGMOD Record*, 18(3):5–11, 1989.

[SK91] Michael Stonebraker and Greg Kemnitz. The POSTGRES next-generation database management system. *Communications of the ACM*, 34(10), October 1991.

[SM02] Andreas Schaad and Jonathan D. Moffett. A lightweight approach to specification and analysis of role-based access control extensions. In *Proceedings of the Seventh ACM Symposium on Access Control Models And Technologies*, pages 13–22. ACM Press, 2002.

[SMB03] David Scott, Alan Mycroft, and Alastair Beresford. Spatial policies for sentient mobile applications. In *Policy 2003: IEEE Fourth International Workshop on Policies for Distributed Systems and Networks*, 2003.

[SMJ01] Andreas Schaad, Jonathan D. Moffett, and Jeremy Jacob. The role-based access control system of a European bank: a case study and discussion. In *Proceedings of the Sixth ACM Symposium on Access Control Models And Technologies*, pages 3–9. ACM Press, 2001.

[SS94] Ravi S. Sandhu and Pierrangela Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

[Sun96] Sun Microsystems. The Java Servlet API White Paper. `http://java.sun.com/products/servlet/whitepaper.html`, 1996.

[Sun99] Sun Microsystems. JavaServer Pages Technology White Paper. `http://java.sun.com/products/jsp/whitepaper.html`, 1999.

[SZ97] Richard T. Simon and Mary E. Zurko. Separation of duty in role-based environments. In *PCSFW: Proceedings of the Tenth Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[TBB$^+$04] Robert Tansley, Mick Bass, Margret Branschofsky, Grace Carpenter, Greg McClellan, and David Stuve. DSpace System Documentation. `http://dspace.org/technology/system-docs/`, August 2004.

[Tho97] Roshan Thomas. Team-based access control (TMAC). In *Proceedings of the Second ACM Workshop on Role-Based Access Control (RBAC-97)*, pages 13–22, New York, November 6–7 1997. ACM Press.

[TIB99] TIBCO. Tibco rendezvous. `http://www.rv.tibco.com`, 1999.

[Tom99] Dimitrios Tombros. *An Event- and Repository-based Component Framework for Workflow System Architecture*. PhD thesis, University of Zurich, 1999.

[vdA96] Wil M.P. van der Aalst. Three good reasons for using a Petri-net-based workflow management system. In Shamkant Navathe and Toshiro Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Cambridge, Massachusetts, November 1996.

[VLD02] VLDB Endowment. *Twenty-Eighth International Conference on Very Large Data Bases VLDB*, Hong Kong, China, August 2002.

[W3C99] W3C. Resource description framework (RDF) model and syntax specification. `http://www.w3.org/TR/rdf-primer/`, February 1999.

[W3C00] W3C. Extensible Markup Language (XML) 1.0 (second edition). `http://www.w3.org/TR/REC-xml`, October 2000.

[W3C01a] W3C. Scalable vector graphics (SVG) 1.0 specification. `http://www.w3.org/TR/SVG/`, September 2001.

[W3C01b] W3C. XML Schema part 0: Primer. `http://www.w3.org/TR/xmlschema-0/`, May 2001.

[W3C01c] W3C. XML Schema part 1: Structures. `http://www.w3.org/TR/xmlschema-1/`, May 2001.

[W3C01d] W3C. XML Schema part 2: Datatypes. `http://www.w3.org/TR/xmlschema-2/`, May 2001.

[W3C03a] W3C. SOAP version 1.2 part 0: Primer. `http://www.w3.org/TR/soap12-part0/`, June 2003.

[W3C03b] W3C. SOAP version 1.2 part 1: Messaging framework. `http://www.w3.org/TR/soap12-part1/`, June 2003.

[W3C03c] W3C. SOAP version 1.2 part 2: Adjuncts. `http://www.w3.org/TR/soap12-part2/`, June 2003.

[W3C04] W3C. Extensible markup language (XML) 1.0 (third edition). `http://www.w3.org/TR/REC-xml/`, February 2004.

[WBK01] Jacques Wainer, Paulo Barthelmess, and Akhil Kumar. W-RBAC - a workflow security model incorporating controlled overriding of constraints. Technical Report IC-01-13, Institute of Computing, University of Campinas, Brazil, October 2001.

[WCEW02] Chenxi Wang, Antonio Carzaniga, David Evans, and Alexander L. Wolf. Security issues and requirements in internet-scale publish-subscribe systems. In *Proceedings of the Thirty-Fifth Annual Hawaii International Conference on System Sciences (HICSS'02)*, 2002.

[Web97] WebCT Educational Technologies Corp. WebCT. `http://www.webct.com/`, September 1997.

[WfM95] Workflow Management Coalition (WfMC). The Workflow Reference Model: Issue 1.1. `www.wfmc.org/standards/docs.htm`, 1995.

[Wie87] Jan Wielemaker. SWI Prolog. `http://www.swi-prolog.org/`, 1987.

[Wie01] Jan Wielemaker. SWI Prolog SGML/XML parser. `http://www.swi.psy.uva.nl/projects/SWI-Prolog/packages/sgml/sgml2pl.html`, March 2001.

[WJH97] Andy Ward, Alan Jones, and Andy Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, October 1997.

[WKB01] Jacques Wainer, Akhil Kumar, and Paulo Barthelmess. Security management in the presence of delegation and revocation in workflow systems. Technical Report IC-01-14, Institute of Computing, University of Campinas, Brazil, October 2001.

[WL02] William H. Winsborough and Ninghui Li. Towards practical automated trust negotiation. In *Procedings of the Third International Workshop on Policies for Distributed Systems and Networks*, Monterey, California, U.S.A., June 2002.

[Yao02] Walt Yao. *Trust Management for Widely Distributed Systems.* PhD thesis, University of Cambridge Computer Laboratory, 2002.

[YMB01] Walt Yao, Ken Moody, and Jean M. Bacon. A model of OASIS role-based access control and its support for active security. In *Procedings of the Sixth ACM Symposium on Access Control Models and Technologies*, pages 171–181, 2001.

[YWS01] Ting Yu, Marianne Winslett, and Kent E. Seamons. Interoperable strategies in automated trust negotiation. In *Proceedings of the Eighth ACM conference on Computer and Communications Security*, pages 146–155. ACM Press, 2001.

[ZOS03] Xinwen Zhang, Sejong Oh, and Ravi S. Sandhu. PDBM: a flexible delegation model in RBAC. In *Procedings of the Eighth ACM Symposium on Access Control Models and Technologies*, 2003.

[ZPPPS04] Xinwen Zhang, Jaehong Park, Francesco Parisi-Presicce, and Ravi S. Sandhu. A logical specification for usage control. In *Proceedings of the Ninth ACM Symposium on Access Control Models And Technologies*, pages 1–10. ACM Press, 2004.

A Full diagnostic output and selected source code

This appendix provides complete transcripts of the diagnostic output from test runs and selected source code listings discussed in chapter 7.

Note that whitespace modifications have been made to emphasise the structure of the underlying text. Also some coloured syntax highlighting has been applied.

A.1 Location data access control listings

In the subsections below we provide material relevant to the access control of location data experiment described in section 7.3.

A.1.1 XML policy file used for location aware access control test

```
<?xml version="1.0"?>
<!DOCTYPE policies PUBLIC "//EDSAC//DTD policies//EN"
                          "http://www.cl.cam.ac.uk/~dme26/EDSAC21/policy3.dtd" >

<policies>

  <definitions>
    <role name="generalLocator"/>
    <role name="knownLocator"/>

    <privilege name="coarseLocation">
      <output name="id" type="prolog.atom" />
    </privilege>

    <privilege name="fineLocation">
      <output name="id" type="prolog.atom" />
    </privilege>

  </definitions>

  <rules>

    <activation role="generalLocator"/>

    <activation role="knownLocator">
      <condition name="generalLocator">
    </activation>

    <authorization privilege="coarseLocation">
      <authorizing-role name="generalLocator">
        <with name="id" value="$id" />
      </authorizing-role>
    </authorization>

    <authorization privilege="fineLocation">
      <authorizing-role name="knownLocator">
        <with name="id" value="$id" />
      </authorizing-role>
    </authorization>

  </rules>
</policies>
```

A.1.2 Requests made by Alice of Bob's location

Time	Action
3	printRoles
3	actionReq(user,9000,default,privilege(coarseLocation,v1,9000,default,[1]))
3	actionReq(user,9000,default,privilege(fineLocation,v1,9000,default,[1]))
4	actionReq(user,9000,default,role(generalLocator,v1,9000,default,[],fact))
4	printRoles
4	actionReq(user,9000,default,privilege(coarseLocation,v1,9000,default,[1]))
4	actionReq(user,9000,default,privilege(fineLocation,v1,9000,default,[1]))
5	actionReq(user,9000,default,role(knownLocator,v1,9000,default,[],fact))
5	printRoles
5	actionReq(user,9000,default,privilege(coarseLocation,v1,9000,default,[1]))
5	actionReq(user,9000,default,privilege(fineLocation,v1,9000,default,[1]))

A.1.3 Diagnostic output from Bob's EDSAC node

```
:::[3] Active roles in this database: [].
-->[3] subscription call-back event:
       <actionReq(user, 9000, default,
            privilege(coarseLocation, v1, 9000, default, [1]))>
-->[3] subscription call-back event:
       <actionReq(user, 9000, default,
            privilege(fineLocation, v1, 9000, default, [1]))>
-->[4] subscription call-back event:
       <actionReq(user, 9000, default,
            role(generalLocator, v1, 9000, default, [], fact))>
<--[4] publishing event:
       <roleActConfirm(_G373, _G374, default,
            role(generalLocator, v1, 9000, default, [], fact))>.
:::[4] Active roles in this database: [
            role(generalLocator, v1, 9000, default, [], fact)].
-->[4] subscription call-back event:
       <actionReq(user, 9000, default,
            privilege(coarseLocation, v1, 9000, default, [1]))>
<--[4] publishing event:
       <privVote(src, _G362, default,
            privilege(coarseLocation, v1, 9000, default, [1]), 1)>.
-->[4] subscription call-back event:
       <actionReq(user, 9000, default,
            privilege(fineLocation, v1, 9000, default, [1]))>
-->[5] subscription call-back event:
       <actionReq(user, 9000, default,
            role(knownLocator, v1, 9000, default, [], fact))>
<--[5] publishing event:
       <roleActConfirm(_G373, _G374, default,
            role(knownLocator, v1, 9000, default, [], fact))>.
:::[5] Active roles in this database: [
            role(generalLocator, v1, 9000, default, [], fact),
            role(knownLocator, v1, 9000, default, [], fact)].
-->[5] subscription call-back event:
       <actionReq(user, 9000, default,
            privilege(coarseLocation, v1, 9000, default, [1]))>
<--[5] publishing event:
       <privVote(src, _G362, default,
            privilege(coarseLocation, v1, 9000, default, [1]), 1)>.
-->[5] subscription call-back event:
       <actionReq(user, 9000, default,
            privilege(fineLocation, v1, 9000, default, [1]))>
<--[5] publishing event:
       <privVote(src, _G362, default,
            privilege(fineLocation, v1, 9000, default, [1]), 1)>.
```

A.1.4 Diagnostic output from the location management node

```
-->[4] subscription call-back event:
       <privVote(src, 9001, default,
            privilege(coarseLocation, v1, 9000, default, [1]), 1)>
:::[4] Location of principal <1> is <151.191, -33.8664> with error 1.
-->[5] subscription call-back event:
       <privVote(src, 9001, default,
            privilege(coarseLocation, v1, 9000, default, [1]), 1)>
:::[5] Location of principal <1> is <151.191, -33.8664> with error 1.
-->[5] subscription call-back event:
       <privVote(src, 9001, default,
            privilege(fineLocation, v1, 9000, default, [1]), 1)>
:::[5] Location of principal <1> is <151.165, -33.9379> with error 5e-05.
```

A.2 Workflow listings

Below we provide the raw data which was used to generate the workflow experiment summary tables discussed in section 7.4.

A.2.1 Commands issued to EDSAC nodes

Time	Target	Action
1	C	fullContexts
1	C	latchedContexts
1	all	printRoles
1	A	actionReq(user, 9000, default, role(wf_transformData, v1, 9000, default, [wf(1), instance(1)], fact))
1	A	'termTrue(role(wf_checkData, v1, 9000, wfc_transformData, [wf(1), instance(1)], rule))'
1	A	actionReq(user, 9000, wfc_transformData, role(wf_checkData, v1, 9000, wfc_transformData, [wf(1), instance(1)], fact))
6	C	fullContexts
6	all	printRoles
6	A	'assert(environment(emulateConditions, v1, 9000, default, []))'
6	A	actionReq(user, 9000, wfc_transformData, role(wf_checkData, v1, 9000, wfc_transformData, [wf(1), instance(1)], fact))
11	C	fullContexts
11	all	printRoles
11	A	actionReq(user, 9000, wfc_checkData, role(wf_checkOrderType, v1, 9000, wfc_checkData, [wf(1), instance(1)], fact))
16	A	actionReq(user, 9000, wfc_checkOrderType, role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType, [wf(1), instance(1)], fact))
21	A	actionReq(user, 9000, default, privilege(delegateCheckVendorAccount, v1, 9000, wfc_checkOrderType, [9001, wf(1), instance(1)]))
26	A	actionReq(user, 9000, default, role(clientLiaison, v1, 9000, default, [], fact))
31	B	actionReq(user, 9001, default, role(clientLiaison, v1, 9001, default, [], fact))
31	A	credRevoke(1, 9000, default, role(clientLiaison, v1, 9000, default, [], fact))
31	A	actionReq(user, 9000, wfc_checkVendorAccount, role(wf_enterOrder, v1, 9000, wfc_checkVendorAccount, [wf(1), instance(1)], fact))
36	C	fullContexts
36	C	latchedContexts
36	all	printRoles
36	B	actionReq(user, 9001, wfc_checkVendorAccount, role(wf_composeRejectMsg, v1, 9001, wfc_checkVendorAccount, [wf(1), instance(1)], fact))

36	B	actionReq(user, 9001, wfc_checkVendorAccount, role(wf_enterOrder, v1, 9001, wfc_checkVendorAccount, [wf(1), instance(1)], fact))
41	B	actionReq(user, 9001, default, role(wf_fillOrder, v1, 9001, default, [wf(1), instance(1)], fact))
41	A	actionReq(user, 9000, default, role(wf_composeAcceptance, v1, 9000, default, [wf(1), instance(1)], fact))
41	A	actionReq(user, 9000, default, role(wf_emailConfirmation, v1, 9000, default, [wf(1), instance(1)], fact))
41	A	'termTrue(role(wf_cleanUp, v1, 9000, default, [wf(1), instance(1)], rule))'
41	B	actionReq(user, 9001, default, privilege(delegateFillOrder, v1, 9001, default, [9000, wf(1), instance(1)]))
41	A	actionReq(user, 9000, default, role(wf_cleanUp, v1, 9000, default, [wf(1), instance(1)], fact))
41	C	fullContexts
41	C	latchedContexts
41	all	printRoles
41	A	actionReq(user, 9000, default, privilege(wf_finish, v1, 9000, default, [wf(1), instance(1)]))

A.2.2 Output from context manager

```
::: [1] Full contexts: [].
::: [1] Context latches: [].
::: [1] Active roles in this database: [].
::: [6] Full contexts: [].
::: [6] Active roles in this database: [].
--> [6] subscription call-back event:
        <roleActReq(_G159, 9016, wfc_transformData,
            role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], fact), 10)>
-->[10] subscription call-back event:
        <roleActConfirm(_G159, 9016, wfc_transformData,
            role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], fact))>
:::[11] Full contexts: [wfc_transformData].
:::[11] Active roles in this database: [].
-->[11] subscription call-back event:
        <roleActReq(_G159, 9016, wfc_checkData,
            role(wf_checkOrderType, v1, 9000, wfc_checkData,
                [wf(1), instance(1)], fact), 15)>
-->[15] subscription call-back event:
        <roleActConfirm(_G159, 9016, wfc_checkData,
            role(wf_checkOrderType, v1, 9000, wfc_checkData,
                [wf(1), instance(1)], fact))>
-->[16] subscription call-back event:
        <roleActReq(_G159, 9016, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType,
                [wf(1), instance(1)], fact), 20)>
```

```
-->[20] subscription call-back event:
        <roleActConfirm(_G159, 9016, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType,
                [wf(1), instance(1)], fact))>
-->[21] subscription call-back event:
        <roleActReq(_G159, 9016, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9001, wfc_checkOrderType,
                [wf(1), instance(1)], fact), 25)>
-->[25] subscription call-back event:
        <roleActConfirm(_G159, 9016, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9001, wfc_checkOrderType,
                [wf(1), instance(1)], fact))>
-->[26] subscription call-back event:
        <roleActReq(_G149, 9016, default,
            role(clientLiaison, v1, 9000, default, [], fact), 30)>
-->[31] subscription call-back event:
        <roleActReq(_G149, 9016, default,
            role(clientLiaison, v1, 9001, default, [], fact), 35)>
-->[31] subscription call-back event:
        <roleActVote(_G149, 9016, default,
            role(clientLiaison, v1, 9001, default, [], fact), -1)>
-->[31] subscription call-back event:
        <roleActReq(_G159, 9016, wfc_checkVendorAccount,
            role(wf_enterOrder, v1, 9000, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact), 35)>
-->[35] subscription call-back event:
        <roleActConfirm(_G159, 9016, wfc_checkVendorAccount,
            role(wf_enterOrder, v1, 9000, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact))>
:::[36] Full contexts: [wfc_transformData, wfc_checkData].
:::[36] Context latches: [
            cl(wfc_checkOrderType, role(wf_checkVendorAccount,
                 v1, _G192, wfc_checkOrderType, _G194, fact)),
            cl(wfc_checkVendorAccount, role(wf_enterOrder,
                 v1, _G176, wfc_checkVendorAccount, _G178, fact))].
:::[36] Active roles in this database: [].
-->[36] subscription call-back event:
        <roleActReq(_G159, 9016, wfc_checkVendorAccount,
            role(wf_composeRejectMsg, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact), 40)>
<--[36] publishing event:
        <roleActVote(_G237, _G238, wfc_checkVendorAccount,
            role(wf_composeRejectMsg, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact), -1)>.
-->[36] subscription call-back event:
        <roleActReq(_G159, 9016, wfc_checkVendorAccount,
            role(wf_enterOrder, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact), 40)>
-->[40] subscription call-back event:
        <roleActConfirm(_G159, 9016, wfc_checkVendorAccount,
            role(wf_enterOrder, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact))>
:::[41] Full contexts: [wfc_transformData, wfc_checkData].
:::[41] Context latches: [
            cl(wfc_checkOrderType, role(wf_checkVendorAccount,
                 v1, _G192, wfc_checkOrderType, _G194, fact)),
            cl(wfc_checkVendorAccount, role(wf_enterOrder,
                 v1, _G176, wfc_checkVendorAccount, _G178, fact))].
:::[41] Active roles in this database: [].
```

A.2.3 Output from node ‘A’

```
::: [1] Active roles in this database: [].
--> [1] subscription call-back event:
        <actionReq(user, 9000, default,
            role(wf_transformData, v1, 9000, default,
                [wf(1), instance(1)], fact))>
<-- [1] publishing event:
        <roleActConfirm(_G405, _G406, default,
            role(wf_transformData, v1, 9000, default,
                [wf(1), instance(1)], fact))>.
::: [1] Term
        <role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], rule)> evaluates: false.
--> [1] subscription call-back event:
        <actionReq(user, 9000, wfc_transformData,
            role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], fact))>
::: [6] Active roles in this database: [
            role(wf_transformData, v1, 9000, default,
                [wf(1), instance(1)], fact)].
--> [6] subscription call-back event:
        <actionReq(user, 9000, wfc_transformData,
            role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], fact))>
<-- [6] publishing event:
        <roleActReq(_G420, _G421, wfc_transformData,
            role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], fact), 10)>.
--- Evaluating term:
        <activateCallback(8,
            role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], fact))>.
<--[10] publishing event:
        <roleActConfirm(_G342, _G343, wfc_transformData,
            role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], fact))>.
:::[11] Active roles in this database: [
            role(wf_transformData, v1, 9000, default,
                [wf(1), instance(1)], fact),
            role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], fact)].
-->[11] subscription call-back event:
        <actionReq(user, 9000, wfc_checkData,
            role(wf_checkOrderType, v1, 9000, wfc_checkData,
                [wf(1), instance(1)], fact))>
<--[11] publishing event:
        <roleActReq(_G420, _G421, wfc_checkData,
            role(wf_checkOrderType, v1, 9000, wfc_checkData,
                [wf(1), instance(1)], fact), 15)>.
--- Evaluating term:
        <activateCallback(12,
            role(wf_checkOrderType, v1, 9000, wfc_checkData,
                [wf(1), instance(1)], fact))>.
<--[15] publishing event:
        <roleActConfirm(_G342, _G343, wfc_checkData,
            role(wf_checkOrderType, v1, 9000, wfc_checkData,
                [wf(1), instance(1)], fact))>.
-->[16] subscription call-back event:
        <actionReq(user, 9000, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType,
                [wf(1), instance(1)], fact))>
<--[16] publishing event:
        <roleActReq(_G420, _G421, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType,
                [wf(1), instance(1)], fact), 20)>.
--- Evaluating term:
        <activateCallback(14,
            role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType,
                [wf(1), instance(1)], fact))>.
<--[20] publishing event:
        <roleActConfirm(_G342, _G343, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType,
                [wf(1), instance(1)], fact))>.
-->[21] subscription call-back event:
        <actionReq(user, 9000, default,
            privilege(delegateCheckVendorAccount, v1, 9000,
                wfc_checkOrderType, [9001, wf(1), instance(1)]))>
<--[21] publishing event:
        <credForward(src, 9001, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType,
                [wf(1), instance(1)], fact))>.
<--[21] publishing event:
        <privVote(src, _G573, wfc_checkOrderType,
            privilege(delegateCheckVendorAccount, v1, 9000,
                wfc_checkOrderType, [9001, wf(1), instance(1)]), 1)>.
-->[26] subscription call-back event:
        <actionReq(user, 9000, default,
            role(clientLiaison, v1, 9000, default, [], fact))>
<--[26] publishing event:
        <roleActReq(_G392, _G393, default,
            role(clientLiaison, v1, 9000, default, [], fact), 30)>.
```

```
--- Evaluating term:
        <activateCallback(42,
            role(clientLiaison, v1, 9000, default, [], fact))>.
<--[30] publishing event:
        <roleActConfirm(_G324, _G325, default,
            role(clientLiaison, v1, 9000, default, [], fact))>.
-->[31] subscription call-back event:
        <roleActReq(_G149, 9000, default,
            role(clientLiaison, v1, 9001, default, [], fact), 35)>
<--[31] publishing event:
        <roleActVote(_G211, _G212, default,
            role(clientLiaison, v1, 9001, default, [], fact), -1)>.
-->[31] subscription call-back event:
        <credRevoke(1, 9000, default,
            role(clientLiaison, v1, 9000, default, [], fact))>
-->[31] subscription call-back event:
        <actionReq(user, 9000, wfc_checkVendorAccount,
            role(wf_enterOrder, v1, 9000, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact))>
<--[31] publishing event:
        <roleActReq(_G420, _G421, wfc_checkVendorAccount,
            role(wf_enterOrder, v1, 9000, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact), 35)>.
--- Evaluating term:
        <activateCallback(20,
            role(wf_enterOrder, v1, 9000, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact))>.
<--[35] publishing event:
        <roleActConfirm(_G342, _G343, wfc_checkVendorAccount,
            role(wf_enterOrder, v1, 9000, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact))>.
:::[36] Active roles in this database: [
            role(wf_transformData, v1, 9000, default,
                [wf(1), instance(1)], fact),
            role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], fact),
            role(wf_checkOrderType, v1, 9000, wfc_checkData,
                [wf(1), instance(1)], fact),
            role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType,
                [wf(1), instance(1)], fact),
            role(wf_enterOrder, v1, 9000, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact)].
-->[36] subscription call-back event:
        <roleActVote(_G159, 9000, wfc_checkVendorAccount,
            role(wf_composeRejectMsg, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact), -1)>
-->[41] subscription call-back event:
        <actionReq(user, 9000, default,
            role(wf_composeAcceptance, v1, 9000, default,
                [wf(1), instance(1)], fact))>
<--[41] publishing event:
        <roleActConfirm(_G411, _G412, default,
            role(wf_composeAcceptance, v1, 9000, default,
                [wf(1), instance(1)], fact))>.
-->[41] subscription call-back event:
        <actionReq(user, 9000, default,
            role(wf_emailConfirmation, v1, 9000, default,
                [wf(1), instance(1)], fact))>
<--[41] publishing event:
        <roleActConfirm(_G411, _G412, default,
            role(wf_emailConfirmation, v1, 9000, default,
                [wf(1), instance(1)], fact))>.
:::[41] Term
        <role(wf_cleanUp, v1, 9000, default,
                [wf(1), instance(1)], rule)> evaluates: false.
-->[41] subscription call-back event:
        <credForward(src, 9000, default,
            role(wf_fillOrder, v1, 9001, default,
                [wf(1), instance(1)], fact))>
<--[41] publishing event:
        <roleActConfirm(_G427, _G428, default,
            role(wf_fillOrder, v1, 9000, default,
                [wf(1), instance(1)], fact))>.
-->[41] subscription call-back event:
        <actionReq(user, 9000, default,
            role(wf_cleanUp, v1, 9000, default,
                [wf(1), instance(1)], fact))>
<--[41] publishing event:
        <roleActConfirm(_G423, _G424, default,
            role(wf_cleanUp, v1, 9000, default,
                [wf(1), instance(1)], fact))>.
:::[41] Active roles in this database: [
            role(wf_transformData, v1, 9000, default,
                [wf(1), instance(1)], fact),
            role(wf_checkData, v1, 9000, wfc_transformData,
                [wf(1), instance(1)], fact),
            role(wf_checkOrderType, v1, 9000, wfc_checkData,
                [wf(1), instance(1)], fact),
            role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType,
                [wf(1), instance(1)], fact),
            role(wf_enterOrder, v1, 9000, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact),
            role(wf_composeAcceptance, v1, 9000, default,
                [wf(1), instance(1)], fact),
            role(wf_emailConfirmation, v1, 9000, default,
                [wf(1), instance(1)], fact),
            role(wf_fillOrder, v1, 9000, default,
                [wf(1), instance(1)], fact),
            role(wf_cleanUp, v1, 9000, default,
                [wf(1), instance(1)], fact)].
-->[41] subscription call-back event:
        <actionReq(user, 9000, default,
            privilege(wf_finish, v1, 9000, default,
                [wf(1), instance(1)]))>
<--[41] publishing event:
        <privVote(src, _G382, default,
            privilege(wf_finish, v1, 9000, default,
                [wf(1), instance(1)]), 1)>.
```

A.2.4 Output from node ‘B’

```
::: [1] Active roles in this database: [].
::: [6] Active roles in this database: [].
:::[11] Active roles in this database: [].
-->[21] subscription call-back event:
        <credForward(src, 9001, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9000, wfc_checkOrderType,
                [wf(1), instance(1)], fact))>
<--[21] publishing event:
        <roleActReq(_G436, _G437, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9001, wfc_checkOrderType,
                [wf(1), instance(1)], fact), 25)>.
--- Evaluating term:
        <activateCallback(14,
            role(wf_checkVendorAccount, v1, 9001, wfc_checkOrderType,
                [wf(1), instance(1)], fact))>.
<--[25] publishing event:
        <roleActConfirm(_G336, _G337, wfc_checkOrderType,
            role(wf_checkVendorAccount, v1, 9001, wfc_checkOrderType,
                [wf(1), instance(1)], fact))>.
-->[26] subscription call-back event:
        <roleActReq(_G149, 9001, default,
            role(clientLiaison, v1, 9000, default, [], fact), 30)>
-->[31] subscription call-back event:
        <actionReq(user, 9001, default,
            role(clientLiaison, v1, 9001, default, [], fact))>
<--[31] publishing event:
        <roleActReq(_G392, _G393, default,
            role(clientLiaison, v1, 9001, default, [], fact), 35)>.
-->[31] subscription call-back event:
        <roleActVote(_G149, 9001, default,
            role(clientLiaison, v1, 9001, default, [], fact), -1)>
:::[36] Active roles in this database: [
            role(wf_checkVendorAccount, v1, 9001, wfc_checkOrderType,
                [wf(1), instance(1)], fact)].
-->[36] subscription call-back event:
        <actionReq(user, 9001, wfc_checkVendorAccount,
            role(wf_composeRejectMsg, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact))>
<--[36] publishing event:
        <roleActReq(_G420, _G421, wfc_checkVendorAccount,
            role(wf_composeRejectMsg, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact), 40)>.
-->[36] subscription call-back event:
        <roleActVote(_G159, 9001, wfc_checkVendorAccount,
            role(wf_composeRejectMsg, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact), -1)>
-->[36] subscription call-back event:
        <actionReq(user, 9001, wfc_checkVendorAccount,
            role(wf_enterOrder, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact))>
<--[36] publishing event:
        <roleActReq(_G420, _G421, wfc_checkVendorAccount,
            role(wf_enterOrder, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact), 40)>.
--- Evaluating term:
        <activateCallback(20,
            role(wf_enterOrder, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact))>.
<--[40] publishing event:
        <roleActConfirm(_G342, _G343, wfc_checkVendorAccount,
            role(wf_enterOrder, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact))>.
-->[41] subscription call-back event:
        <actionReq(user, 9001, default,
            role(wf_fillOrder, v1, 9001, default,
                [wf(1), instance(1)], fact))>
<--[41] publishing event:
        <roleActConfirm(_G411, _G412, default,
            role(wf_fillOrder, v1, 9001, default,
                [wf(1), instance(1)], fact))>.
-->[41] subscription call-back event:
        <actionReq(user, 9001, default,
            privilege(delegateFillOrder, v1, 9001, default,
                [9000, wf(1), instance(1)]))>
<--[41] publishing event:
        <credForward(src, 9000, default,
            role(wf_fillOrder, v1, 9001, default,
                [wf(1), instance(1)], fact))>.
<--[41] publishing event:
        <privVote(src, _G573, default,
            privilege(delegateFillOrder, v1, 9001, default,
                [9000, wf(1), instance(1)]), 1)>.
:::[41] Active roles in this database: [
            role(wf_checkVendorAccount, v1, 9001, wfc_checkOrderType,
                [wf(1), instance(1)], fact),
            role(wf_enterOrder, v1, 9001, wfc_checkVendorAccount,
                [wf(1), instance(1)], fact),
            role(wf_fillOrder, v1, 9001, default,
                [wf(1), instance(1)], fact)].
```