

Pure

Bond University

DOCTORAL THESIS

Secure information flow for inter-organisational collaborative environments

Bracher, Shane

Award date:
2009

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 10. May. 2019



Secure Information Flow for Inter-organisational Collaborative Environments

Shane Bracher

BIT, MIT (Hons) (Bond)

A dissertation submitted in fulfilment of the requirements of the degree of
Doctor of Philosophy for the School of Information Technology, Bond University.

April 2009

Copyright © 2009 Shane Bracher

Typeset in L^AT_EX 2_ε

Statement of Originality

The work presented in this thesis is, to the best of my knowledge and belief, original, except where acknowledged in the text. I hereby declare that I have not submitted this material either in whole or in part, for a degree at this or any other university.

Shane Bracher

Date:

Submitted for examination: August 2008

Approved for the degree of Doctor of Philosophy: April 2009

Abstract

Collaborative environments allow users to share and access data across networks spanning multiple administrative domains and beyond organisational boundaries. This poses several security concerns such as data confidentiality, data privacy and threats to improper data usage. Traditional access control mechanisms focus on centralised systems and implicitly assume that all resources reside in the one domain. This serves as a critical limitation for inter-organisational collaborative environments, which are characteristically decentralised, distributed and heterogeneous.

A consequence of the lack of suitable access control mechanisms for inter-organisational collaborative environments is that data owners relinquish all control over data they release. In these environments, we can reasonably consider more complex cases where documents may have multiple contributors, all with differing access control requirements. Facilitating such cases, as well as maintaining control over the document's content, its structure and its flow path as it circulates through multiple administrative domains, is a non-trivial issue.

This thesis proposes an architecture model for specifying and enforcing access control restrictions on sensitive data that follows a pre-defined inter-organisational workflow. Our approach is to embed access control enforcement within the workflow object (e.g. the circulating document containing sensitive data) as opposed to relying on each administrative domain to enforce the access control policies. The architecture model achieves this using cryptographic access control – a concept that relies on cryptography to enforce access control policies.

The specifications for the architecture model, as well as the specifications for an workflow object model, are discussed in this thesis. The workflow object model supports finer-granularity access control (on the content level) and describes how the workflow object encapsulates sensitive data, together with metadata defining the access rights. A prototype implementation of the models was constructed for experimentation purposes. Using this prototype, case studies were conducted to demonstrate the feasibility of the proposed models and to identify potential applications.

Acknowledgements

This thesis would not have been possible without the support of many people and organisations to whom I express my gratitude. In particular, I would like to thank:

Dr Padmanabhan (Paddy) Krishnan, for his supervision, advice, encouragement, enthusiasm and commitment throughout my PhD candidature. Whenever I needed assistance, you were always available to answer my questions, share ideas and offer valuable feedback.

Dr Jorge Cuellar, for his supervision during my internship at Siemens AG (Corporate Technology) and ideas for my research.

Dr Zheng da Wu, for his guidance during the early stages of my PhD candidature and also during my honours research.

The *School of Information Technology, Bond University*, for providing the resources needed to complete my PhD. I also extend my gratitude to all staff and students in the School, past and present, who have helped me and made my time at Bond all the more enjoyable.

The *Australian Government*, for supporting me with an Australian Postgraduate Award.

Fellow PhD candidates and officemates *James Larkin* and *Percy Pari-Salas*, for the discussions, humour, assistance and for creating a great office atmosphere.

My colleagues and officemates at Siemens: *Dr Joerg Abendroth*, *Dong Huang*, *Dr Anja Jerichow*, *Assadarat Khurat*, *Dr Monika Maidl*, *Hariharan Rajasekaran*, *Dr David von Oheimb* and all staff and students who helped make my stay such a pleasant experience.

Finally, my family, for their continuous support and encouragement. Thank you.

Contents

1	Introduction	1
1.1	Objectives	4
1.2	Deliverables	5
1.3	Thesis Structure	6
2	Literature Review	7
2.1	Workflow Management	7
2.1.1	Workflow Concepts	7
2.1.2	Workflow Frameworks	9
2.1.3	Workflow Patterns	12
2.1.4	Workflow Implementation Models	15
2.2	Security Policies and Access Control	18
2.2.1	Security Concepts	18
2.2.2	Security Policies and Models	25
2.2.3	Traditional Access Control Models	33
2.2.4	Implementing Access Control	37
2.2.5	Emerging Access Control Technologies	38
2.3	Access Control for Inter-organisational Workflows	45
2.4	Cryptographic Access Control	46
2.5	Summary	48

3	Architecture Model	49
3.1	Overview	49
3.2	Example Workflow	52
3.3	User-based Model	53
3.4	Role-based Model	54
3.5	Workflow Instantiation	55
3.6	Key Exchange	56
3.6.1	User-defined Shared Keys	58
3.7	Access Types	59
3.7.1	Read Only Access	60
3.7.2	Read Write Access	60
3.7.3	Append Access	61
3.8	Workflow Object History	63
3.9	Multiple Domains	65
3.10	Multiple Permissions	68
3.11	Identity Provider as Verifier	69
3.12	Summary	70
4	Workflow Object Model	71
4.1	Components	71
4.2	Specifications	73
4.2.1	Object Data Component	74
4.2.2	Object Metadata Component	74
4.2.3	Schema Component	75
4.2.4	Policies Component	76
4.2.5	Template Component	77
4.3	Summary	77
5	Prototype Implementation	78
5.1	Overview	78

5.2	Workflow Engine	80
5.3	Identity Provider	82
5.3.1	Identity Management	83
5.3.2	Key Management	84
5.4	Document Processing Environment	86
5.5	Summary	89
6	Case Studies	90
6.1	Case Study 1 – Loan Application	90
6.1.1	Process	91
6.1.2	Document Design	93
6.1.3	Users and Roles	96
6.1.4	Implementation	97
6.1.5	Discussion	101
6.2	Case Study 2 – Electronic Health Records	103
6.2.1	Process	103
6.2.2	Document Design	106
6.2.3	Users and Roles	107
6.2.4	Implementation	108
6.2.5	Discussion	112
6.3	Case Study 3 – Wildlife Report	114
6.3.1	Policies	115
6.3.2	Document Design	115
6.3.3	Implementation	116
6.3.4	Discussion	116
6.4	Lessons Learned	117
6.5	Summary	119
7	Conclusion	121
7.1	Contributions	122

7.2 Future Work	123
A Acronyms	125
B Source Code	128
C Publications	129
C.1 Publications by the Candidate Relevant to the Thesis	129
C.2 Additional Publications by the Candidate Relevant to the Thesis but not Forming Part of it	130

List of Figures

2.1	Workflow reference model.	10
2.2	Workflow perspectives.	11
2.3	Web services base model.	17
2.4	The basic model for conventional encryption.	19
2.5	The basic model for public-key encryption.	20
2.6	Chinese Wall data hierarchy.	26
2.7	Bell-LaPadula model example.	28
2.8	ORCON example.	29
2.9	ORCON with UCON example.	30
2.10	A typical DRM system.	32
2.11	ACL and capability list examples.	35
2.12	The Role-Based Access Control model (Core RBAC).	37
2.13	The SAML producer-consumer model.	40
2.14	The XACML architecture.	42
2.15	The Web Services Security framework.	43
3.1	Simplified view of the architecture model.	51
3.2	Example workflow used for subsequent explanations of the architecture model.	52
3.3	Request/response procedure for obtaining shared key.	58
3.4	Example workflow modified to include multiple domains.	66
3.5	Architecture extension to support verification by identity providers.	69
4.1	The workflow object model components.	72

5.1	Overview of the prototype implementation setup.	79
5.2	Workflow simulation setup for prototype.	81
5.3	Parallel transitions.	81
5.4	Intermediary transitions.	82
5.5	Identity management system setup.	84
5.6	Key management setup for prototype.	85
5.7	Document processing environment setup.	87
6.1	Loan application workflow.	91
6.2	Top-level view of document data component.	94
6.3	Document data component section for processing.	95
6.4	Top-level view of document metadata component.	95
6.5	Protected data items section in document metadata component.	95
6.6	Screenshot of loan application form.	99
6.7	Screenshot of protected data items on form.	100
6.8	Medical record workflow.	104
6.9	Workflow history section in document metadata component.	106
6.10	Top-level view of medical record schema.	106
6.11	Screenshots of verifier interface.	110
6.12	Remodelled medical record workflow.	111
6.13	Modified design for protected data item records.	116

Chapter 1

Introduction

The emergence of the Internet as a common communication medium has attributed to the ever-increasing amounts of data in digital form. We continue to see a proliferation of computing devices capable of sharing and accessing data across networks spanning multiple administrative domains and organisational boundaries. Such collaborative environments pose several security concerns – for instance, risks to data confidentiality, data privacy and threats to improper data usage – leading to increased demands to address these concerns. Hence the shift toward the networking paradigm has required a fundamental re-evaluation of information security and in particular, access control.

Traditional access control mechanisms assign a crucial role to the *reference monitor* [2] to validate access requests. But these mechanisms were originally developed for centralised systems. The implicit assumption that resources reside in one domain may satisfy the access control requirements of intra-organisational scenarios but serves as a critical limitation for inter-organisational environments.

For example, consider the simple case of sending an email with a file attachment to a user in another domain. The access control permissions applied to the file in the originating domain are not retained and enforced in the destination domain. This is because the access control permissions are recognised only within the domain where they were applied. It is therefore clear from this example how inter-organisational environments, characterised by multiple organisations with separate domains, cannot rely on traditional access control mechanisms alone to satisfy their access control requirements.

The ability to share and access data electronically within inter-organisational environments has

attracted much interest in *inter-organisational workflows* [73]. The emergence of *service oriented computing* (a new approach toward distributed computing) has significantly contributed to this interest in recent times. Central to service oriented architectures are *services* – autonomous, platform-independent software components tasked with providing inter-operability and collaboration within heterogeneous environments. Support for this concept has been boosted by industry’s strong embracement of *web services* – the most widely recognised framework of technologies for implementing service oriented architectures.

The aim of web services is a machine-accessible web that enables seamless end-to-end automated processing amongst services. It provides a standards-based, loose-coupling approach for combining multiple services (possibly offered by different organisations) into a single, more sophisticated, value-added, composed service. This composition largely follows the workflow paradigm in that it involves specifying the participating services, the order of interactions (control flow) and the data transferred between services (data flow). As a result, web services are well-suited as an enabling technology for inter-organisational workflows.

An essential requirement of workflows is the ability to transfer documents from one entity to another. From a security perspective, current support for electronic document flow is inadequate for inter-organisational workflows, partly due to the access control limitations for inter-organisational environments.

Of particular interest is the *loss of control* issue. With the exception of some specific office applications (e.g. Adobe PDF), in general when data is released into another administrative domain, the data owner relinquishes all control over it: it can be downloaded, copied, disseminated and re-distributed [46]. A mechanism is needed that suitably allows data owners to maintain control over their data as it flows from one domain to another. Each document may have multiple contributors, and hence multiple data owners, all with differing access control requirements. Maintaining control over the document’s content, its structure and its flow path as it circulates through networks spanning multiple administrative domains is a non-trivial issue.

An example where this is applicable is a *document approval* workflow. Consider a document (assigned to a workflow) that is passed, in sequence, between multiple contributors (or “approvers”) who must each sight and sign the document. The contributors, residing in different domains, may

also add additional data to the document and protect this data with their own access control policies. This example highlights the complications associated with maintaining control over *released* data and serves as the motivation for Case Study 1 of this thesis (discussed in Section 6.1).

The loss of control issue is further challenged by the centralised nature of traditional access control mechanisms and workflow management systems. Inter-organisational collaborative environments cannot rely on the existence of a centralised authority for enforcing security policies [38]. Instead, each domain is separately administrated by its respective organisation. Those involved in an inter-organisational workflow view other participants as simply business partners and nothing else [40]. This leads to security measures being implemented and enforced on a domain (organisation) level rather than on a global (environment) basis.

Using *cryptography* for access control enforcement presents an interesting direction for addressing the access control concerns for inter-organisational environments. Cryptography involves transforming data to hide its information content, to prevent undetected modification or to prevent unauthorised use. Goals include data confidentiality (Section 2.2.1.1), data integrity (Section 2.2.1.2) and non-repudiation (Section 2.2.1.5). It is most commonly used for transmitting data on a channel that lies beyond the trust boundary of a system. With the improvements to cryptographic algorithms, the computational cost of using cryptography for persistent data is greatly reduced. The increased computational power of modern and future computing platforms no longer restrict client devices from applying cryptographic protection.

By revising access control architectures to incorporate cryptographic access control protection, we can potentially eliminate the need for reference monitors [17]. A major advantage of this concept is that the data owner can specify their access control policy and not be involved in its enforcement. Instead, enforcing access control policies would depend on the correct management of shared secrets (keys). This concept also separates the enforcement of access control policies from the storage of protected data. Hence we are not constrained to storing documents (containing protected data) at a certain location. At the same time, we exclusively rely on cryptography to ensure the confidentiality and integrity of the data as it flows throughout the inter-organisational environment.

1.1 Objectives

The main objective of this research is to investigate the issue of access control for inter-organisational collaborative environments. We aim to propose an architecture model that achieves this but greater focus will be applied on its implementation and the case studies. In this respect, the model can be viewed as a practical model. Emphasis is placed on access control restrictions on sensitive data that follows a pre-defined inter-organisational workflow. Effectively, our approach is to embed access control enforcement within the workflow object (e.g. the circulating document containing sensitive data) as opposed to relying on each security realm (administrative domain) to enforce the access control policies. The access control protection itself, depending on how it is applied to the data, implicitly defines the access control restrictions. Hence this is how we intend to achieve embedded access control enforcement.

Another objective is *finer granularity access control* support for the workflow object. We do not assume that the workflow object is a single unit of data. Instead, we model the workflow object as a structured collection of data items. Many access control mechanisms apply restrictions to the whole object (hence affecting all data contained within the object). In our model, we apply access control restrictions to only sensitive data items so that the remainder of the workflow object is unaffected. This finer level of granularity is necessary for collaborative situations where the workflow object may have many contributors from different organisations. It allows the workflow object to be “partitioned” so that contributors can only access the parts of the object that they are authorised to access.

There are considerable challenges associated with these objectives. For a start, the context of this work is not constrained to a single domain with a central authority. It applies to decentralised, distributed, heterogeneous environments. Any solution proposed needs to work within existing Internet frameworks as establishing an entirely new, competing framework would likely introduce excessive costs and would hinder integration efforts.

In terms of access control, key management and distribution is a significant challenge. A strategy is needed that allows authorised users to seamlessly retrieve the keys that provide access to the sensitive data. How keys are created, how they are issued and how they are managed are other issues faced. Securing these keys is critical as a user’s access to the keys effectively determines their ac-

cess privileges on the workflow object. The threat of attacks (e.g. replay attacks, man-in-the-middle attacks, etc.) is also an issue. As a result, different keys are needed for each workflow object instance, including each protected data item within the object. Managing this magnitude of keys poses a challenge.

Flow control is also a challenge. Specifying the workflow rules, tracking the workflow state and verifying the workflow object throughout the workflow are all issues to address. The concept of attaching workflow information and state information to the object not only helps facilitate secure information flow, but it is also a step closer towards realising automated information flow. This potential is greatly influenced by the inclusion of “flow-aware” and “state-aware” capabilities to the workflow object.

These objectives are aimed at addressing the research question of this thesis: How to restrict access to sensitive data following a workflow that spreads across organisational boundaries, in a fashion that is independent from the underlying infrastructure of each organisation, not restricted to any particular implementation software and does not rely on each organisation’s security realm to enforce the access control policies? The deliverables listed in the next section are our proposed solution to this question.

1.2 Deliverables

The deliverables of this thesis include the following:

1. The specifications for an architecture model that secures the flow of sensitive data throughout inter-organisational collaborative environments. This model provides a generic platform for supporting secure information flow and uses cryptographic access control for enforcing the access control policies defined by data owners.
2. The specifications for a workflow object model that defines the components of the object circulated amongst workflow participants. This object encapsulates the sensitive data, together with metadata defining the access control restrictions. The model allows for finer-granularity access control on the content level.
3. The design and construction of a prototype implementation of the architecture model and work-

flow object model for experimentation purposes. This prototype attempts to “plug-in” existing tools where possible.

4. The implementation of several case studies demonstrating potential applications for the proposed models. These case studies use the prototype implementation and report on the issues and experiences encountered.

1.3 Thesis Structure

The subsequent chapters of this thesis are organised as follows: Chapter 2 covers the literature review, summarising related work in this research field. Chapters 3 and 4 present the architecture model and workflow object model respectively. These two models form the proposed solution to the research problem addressed in this thesis. Chapter 5 describes the prototype implementation designed to test the proposed solution. Chapter 6 reports on the case studies conducted to demonstrate potential applications for this research, with Chapter 7 concluding this work.

Chapter 2

Literature Review

This thesis encompasses workflow management, security policies and access control, all in the context of inter-organisational environments. To better understand what these terms mean, this literature review discusses them in detail, covering the main concepts and describing widely embraced models relating to these fields. Implementation-based examples are also included.

As well as covering the relevant background information, this chapter discusses related work based on access control for inter-organisational workflows and cryptographic access control. Although various literature exists in these areas, we take a sample of these academic papers and examine their contributions and their limitations in respect to the objectives of this thesis.

2.1 Workflow Management

We begin the chapter by looking at workflow management. This section discusses the main principles of workflow management and looks at different models that are used to support workflows.

2.1.1 Workflow Concepts

The Workflow Management Coalition (WfMC) defines a *workflow* as:

“The automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal.” [86]

The main purpose of workflows is to *automate* business processes – particularly those processes involving a combination of human and machine-based activities.

Some key workflow concepts, defined in the WfMC Glossary of Terms [85], are listed below:

Business Process – A sequence of activities which collectively pursue a common business objective or policy goal within an organisation or between organisations.

Activity – A discrete process step which may consist of one or more tasks. An activity may be performed manually (requiring human intervention) or automatically (should computer automation be supported).

Instance – A single execution of a process or an activity within a process. An instance also includes any data associated with the process or activity.

Workflow Participant – A resource involved in the workflow instance which carries out the work to execute the process. This work is typically categorised as one or more work items which are allocated to the workflow participant via the worklist.

Work Item – An individual task to be processed by a workflow participant. A work item is represented in the context of an activity.

Worklist – A set of work items which are assigned to a particular workflow participant. In cases where worklists are shared, a group of workflow participants are responsible for executing the set of work items on the worklist.

Early motivation for workflows stemmed from the “paperless office” vision. Whilst the idea of completely eliminating paperwork within office environments never succeeded – in fact, paperwork has since proliferated – isolated office tasks are increasingly becoming automated, hence improving office efficiency.

Efficiency is an important factor, particularly in today’s global economy (characterised by global competition and rapidly changing technology), where an organisation’s level of responsiveness is imperative in order to retain a competitive advantage. Inefficient processes impact an organisation’s ability to react to the demands of business environments.

Although workflows can be manually organised, most workflows are normally organised within the context of a computer-based system. In this case, a *Workflow Management System* (WfMS) is commonly used to provide procedural automation of the business process. A WfMS can be defined as:

“A system that completely defines, manages and executes *workflows* through the execution of software whose order of execution is driven by a computer representation of the workflow logic.” [86]

A WfMS coordinates and controls the execution of workflows through the use of software. It automates business processes by interpreting process definitions, interacting with workflow participants and invoking external software applications. Its primary responsibilities include (1) monitoring and reporting on the performance of processes and the users involved in their execution, (2) enforcing deadlines, (3) ensuring security, and (4) authenticating users [73]. Specific WfMS examples are discussed in [9, 28].

An advantage of these systems is their capacity to separate workflow logic from the logic of other software applications (which are used to execute individual activities in the workflow). This ability allows application programs to operate independently from the WfMS, resulting in simplified enterprise integration.

In the past, a WfMS would mainly focus on automating isolated office procedures. Nowadays, they are shifting towards managing inter-organisational information flows to achieve automation across the entire business process. This shift introduces coordination problems since multiple locations and multiple interests now exist. But this shift is necessary, especially when you consider the current trend of organisations outsourcing more and more of their work, there is a need for the seamless integration of heterogeneous workflow systems across enterprises.

2.1.2 Workflow Frameworks

The Workflow Management Coalition (WfMC) developed the Workflow Reference Model [86] to promote inter-operability between workflow technologies. This model identifies the major components and interfaces associated with workflow systems. Figure 2.1 illustrates the model.

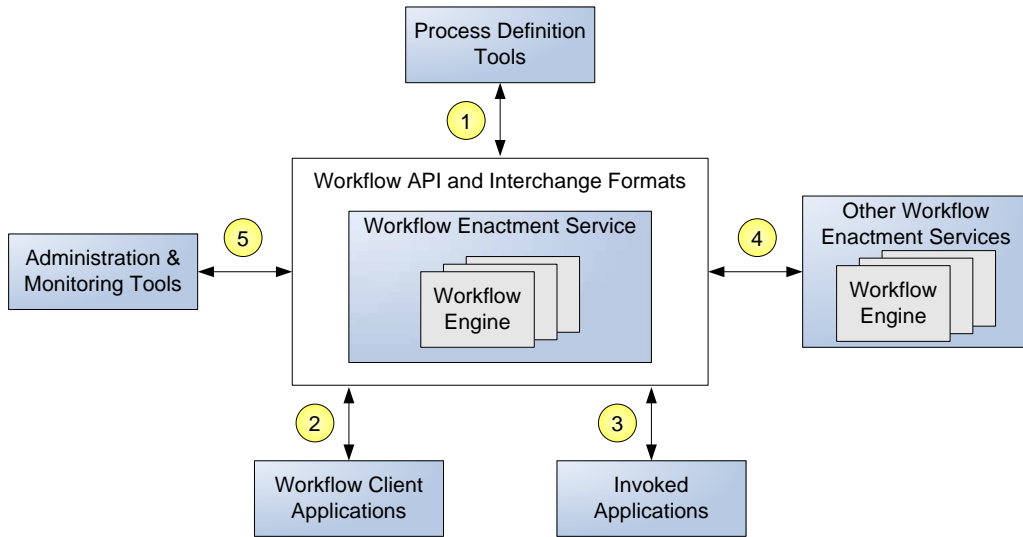


Figure 2.1: Workflow reference model.

The workflow enactment service, shown in the centre of this model, provides the run-time environment where process instantiation occurs, using one or more workflow management engines. External resources interact with this service via the workflow programming interface (WAPI).

Five individual interfaces are specified in the model:

1. Process Definition Tools – used at build-time to define the workflow process.
2. Workflow Client Applications – allows workflow engines to interact with worklist handlers.
3. Invoked Applications – allows workflow engines to interact with user applications (e.g. main-frame legacy systems).
4. Other Workflow Enactment Services – allows workflow engines provided by different vendors to interoperate.
5. Administration and Monitoring Tools – allows interaction between management/monitoring applications and the workflow engines.

The Workflow Reference Model is useful for understanding the relationships between workflow engines, its users and other software systems [73]. But for considering the analysis and design requirements for a workflow system, the framework illustrated in Figure 2.2 is better suited. This framework emphasises five related views or “perspectives” [14, 35].

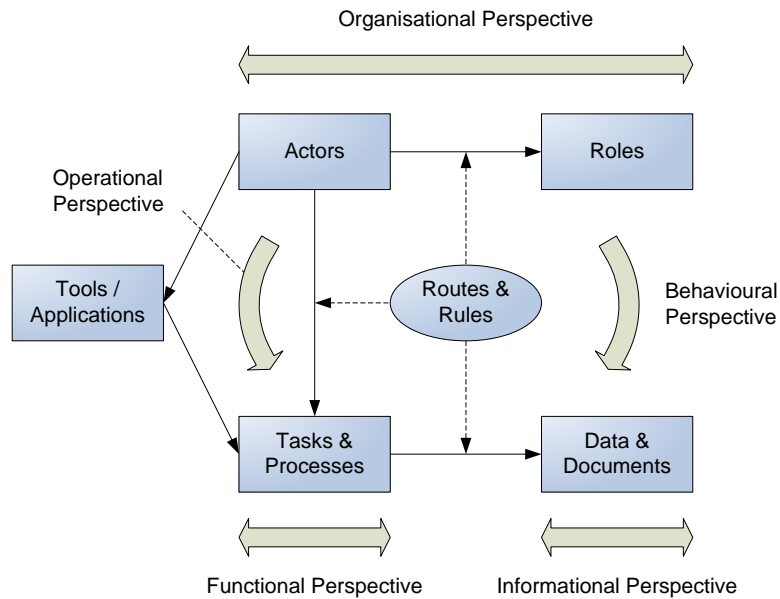


Figure 2.2: Workflow perspectives.

The following list explains each of these perspectives [73]:

- The *functional* perspective – What does the workflow do? Workflow is specified by decomposing high level functions into smaller tasks that can be allocated to users or software agents.
- The *behavioural* perspective – When are the activities and tasks executed? Uses a process model that defines the time precedence of individual activities, events and triggers, and the pre- and post-conditions for activities.
- The *informational* perspective – What data is consumed and produced? Describes the business data, documents and electronic forms that are sent between agents, as well as the files and databases that store persistent application data.
- The *operational* perspective – How is a workflow activity implemented? Specifies the workflow tools and applications that perform the workflow activities.
- The *organisational* perspective – Who performs what tasks and with what tools? Defines the organisational hierarchy, the roles, the security and access authorisations, teams and work groups, and individual users and software applications. A role is a collection of tasks and responsibilities that can be assigned to an agent at run-time.

2.1.3 Workflow Patterns

Workflow patterns representing comprehensive workflow functionality are identified by van der Aalst *et al.* [76]. A pattern can be described as “the abstraction from a concrete form which keeps recurring in specific nonarbitrary contexts” [54]. The purpose of the workflow patterns is to “address business requirements in an imperative workflow style expression ... removed from specific workflow languages” [76].

As argued in [76], although most workflow languages (and workflow management systems for that matter) support the basic workflow constructs, the interpretation of these constructs across different languages is not uniform. Furthermore, it is often unclear how languages even support the more complex workflow constructs.

Workflow patterns are relevant to this thesis as they define workflows on a level of abstraction that is removed from specific implementation tools and workflow languages. This is necessary for the architecture model discussed in Chapter 3 as our research aims to promote inter-operability across different organisations. Hence the architecture model cannot be constrained in respect to workflow definitions, or any other component for that matter.

The following subsections describe the workflow patterns, quoted from [76]. Cook, Patwardhan and Misra [12] provide implementations of these workflow patterns in Orc [47] – a process calculus for orchestrating wide-area computations.

2.1.3.1 Basic Control Flow Patterns

Pattern 1 (Sequence) – “An activity in a workflow process is enabled after the completion of another activity in the same process.”

Pattern 2 (Parallel split) – “A point in the workflow process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order.” Also known as *AND-split*, *parallel routing* and *fork*.

Pattern 3 (Synchronisation) – “A point in the workflow process where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronising multiple threads.” Also known as *AND-join*.

Pattern 4 (Exclusive choice) – “A point in the workflow process where, based on a decision or workflow control data, one of several branches is chosen.” Also known as *XOR-split*, *switch* and *decision*.

Pattern 5 (Simple merge) – “A point in the workflow process where two or more alternative branches come together without synchronisation.” Assumes that no branches are executed in parallel. Also known as *XOR-join*, *asynchronous join* and *merge*.

2.1.3.2 Advanced Branching and Synchronisation Patterns

Pattern 6 (Multi-choice) – “A point in the workflow process where, based on a decision or workflow control data, a number of branches are chosen.” A multi-choice is a non-exclusive choice. Also known as *OR-split*.

Pattern 7 (Synchronising merge) – “A point in the workflow process where multiple paths converge into one single thread. If more than one path is taken, synchronisation of the active threads needs to take place. If only one path is taken, the alternative branches should reconverge without synchronisation.” Also known as *synchronising join*.

Pattern 8 (Multi-merge) – “A point in a workflow process where two or more branches reconverge without synchronisation. If more than one branch gets activated, possibly concurrently, the activity following the merge is started *for every activation of every incoming branch*.”

Pattern 9 (Discriminator) – “The discriminator is a point in a workflow process that waits for one of the incoming branches to complete before activating the subsequent activity.” The completion of all remaining branches is ignored. For example, a search engine may use multiple backend databases, but only the results from the first database to finish executing the search are displayed. The results from the remaining databases are ignored.

2.1.3.3 Structural Patterns

Pattern 10 (Arbitrary cycles) – “A point in a workflow process where one or more activities can be done repeatedly.” Also known as *loop*, *iteration* and *cycle*.

Pattern 11 (Implicit termination) – “A given subprocess should be terminated when there is nothing else to be done. In other words, there are no active activities in the workflow and no other activity can be made active (and at the same time the workflow is not in deadlock).”

2.1.3.4 Patterns Involving Multiple Instances

Pattern 12 (Multiple instances *without* synchronisation) – “Multiple instances of an activity can be created (within a single workflow instance)” In other words, new threads of control can be spawned, executing independently of one another.

Pattern 13 (Multiple instances with a priori *design time* knowledge) – “The number of instances of a given activity for a given (workflow) instance is known at design time. Once all instances are completed, some other activity needs to be started.”

Pattern 14 (Multiple instances with a priori *runtime* knowledge) – “The number of instances of a given activity for a given (workflow instance) varies ... but is known at some stage during runtime, before the instances of that activity have to be created. Once all instances are completed, some other activity needs to be started.”

Pattern 15 (Multiple instances without a priori *runtime* knowledge) – “The number of instances of a given activity for a given (workflow instance) is not known during design time, nor is it known at any stage during runtime, before the instances of that activity have to be created. Once all instances are completed, some other activity needs to be started.”

2.1.3.5 State-Based Patterns

Pattern 16 (Deferred choice) – “A point in the workflow process where one of several branches is chosen (based on information which is not necessarily available when this point is reached). In contrast to the XOR-split, the choice is not made explicitly ... but several alternatives are offered to the environment. Once the environment activates one of the branches, the other alternative branches are withdrawn.”

Pattern 17 (Interleaved parallel routing) – “A set of activities is executed in an arbitrary order: Each activity in the set is executed, the order is decided at runtime and no two activities are

executed at the same moment (i.e. no two activities are active for the same workflow instance at the same time).”

Pattern 18 (Milestone) – “An activity ... is only enabled if a certain milestone has been reached which did not expire yet.”

2.1.3.6 Cancellation Patterns

Pattern 19 (Cancel activity) – “An enabled activity is disabled, i.e. a thread waiting for the executing of an activity is removed.”

Pattern 20 (Cancel case) – “A case (i.e. workflow instance) is removed completely. Even if parts of the process are instantiated multiple times, all descendants are removed.”

2.1.4 Workflow Implementation Models

Three basic architectures exist for implementing workflows [73]:

Production Architecture – Commonly implemented using *workflow folders*. Depending on the implementation, a workflow folder can relate to a particular workflow instance, and this folder (containing all documents related to the workflow instance) is circulated in turn to each workflow participant involved in the workflow. Most existing production architectures consist of a single workflow engine, providing services to users in a client-server architecture fashion.

Messaging-based Architecture – Extend messaging systems (such as internal email systems, web services, etc.) with workflow capabilities (e.g. adding electronic forms, logging and worklist generation capabilities).

Document-centric Architecture – Add workflow capabilities to document management systems. For example, a workflow participant notifies another participant of an activity to be performed. The notified participant checks out the relevant documents from the database, performs the activity, and finally returns the documents to the database for the next participant.

The following sections discuss concrete examples of these implementation approaches.

2.1.4.1 Workflow Folders

This implementation model uses special folders that possess workflow information. These folders act as workflow-aware containers, and depending on the implementation, may be reactive to changes in the workflow state.

X-Folders [57] are an example of where scripts are applied to folders. A change in the document's state (e.g. the document status changing to final) triggers the folder's script into executing some task (e.g. moving the document to another folder).

Another example is the POLITeam project [69]. This project allocates a folder to each workflow. The folder is then circulated amongst the workflow participants, operating on top of a groupware base system. When the folder is received, the participant extracts the required documents from the folder, processes them in accordance to the workflow activity, returns them to the folder, and forwards the folder to the next participant in the workflow.

The POLITeam project example lacks adequate support for inter-organisational workflows. It is better suited for internal workflows within a single organisation (centralised administration authority). Its reliance on an underlying (possibly proprietary) groupware base system hinders inter-operability efforts, unreasonably forcing all partner organisations in the workflow to deploy the same workflow management system.

In contrast, the X-Folders example is inspired by the peer-to-peer model. But the problem with this approach is that it provides no end-to-end workflow description, making it difficult to track the document's global state. Instead, it is better suited for light workflow processes between peers, despite supporting standardised inter-operability technologies.

2.1.4.2 Web Services and Business Process Execution Language

Web services, coupled with Business Process Execution Language (WS-BPEL) [79], provide a promising implementation model for inter-organisational workflows, given its suitability for heterogeneous and Internet-based environments.

Web services can be defined as “an interface that describes a collection of operations that are network-accessible through standardised XML messaging” [31]. They are often viewed as an alternative to traditional middleware solutions such as CORBA, COM+ and EJB. They differ from these

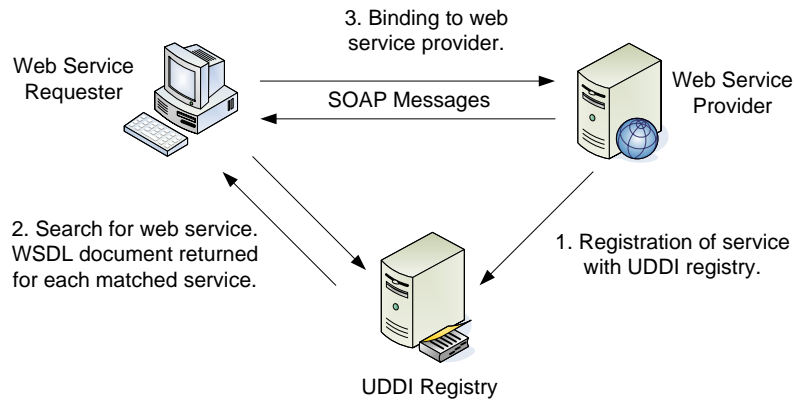


Figure 2.3: Web services base model.

technologies by providing lightweight business process integration. This is possible as web services operate on top of existing Internet protocols such as HTTP.

The Web Services Architecture Stack [90] defines a layered model for illustrating the main components of web services. At the top of this stack is the *Process* layer where WS-BPEL lies. Web services security (discussed later in this chapter) exists as a vertical component in this stack as it affects all layers.

Web Services Base Model When a request for a web service is made, the three step “publish, find, bind” process is followed. This model is illustrated in Figure 2.3.

A web service provider advertises their service in a registry so that others can locate it. Registrations and queries to the registry are performed using the Universal Description, Discovery and Integration Protocol (UDDI) [75]. The information published in the UDDI registry includes the publisher’s name, a description of the service, binding specifications, etc.

A web service requester queries the UDDI registry to find a service. If a matching web service is found, the registry responds with a Web Services Description Language (WSDL) [80] document which details how to bind to this web service. The requester connects to the web service provider by following the specifications in the WSDL document. Finally, the provider responds with the output of the service. Web services messages are communicated via the SOAP protocol [68].

Business Process Execution Language WS-BPEL provides a language for specifying business processes. It extends the web services interaction model by enabling support for business transactions. WS-BPEL defines an interoperable integration model aimed at facilitating automated process integration for both intra-organisational and inter-organisational environments.

Primitive activities supported in WS-BPEL are:

- <invoke> – to invoke an operation of a web service (start an activity).
- <receive> – to wait for an operation to be invoked by another workflow participant.
- <reply> – to generate the response of an operation.
- <assign> – to transfer data from one place to another.

Complex activities supported in WS-BPEL include sequences, branching, loops and parallel execution.

2.2 Security Policies and Access Control

We now shift focus to information security, and in particular, security policies and access control. This section explores various policy and access control models, from abstract policy models to the traditional access control models and finally some emerging access control technologies. But to begin, we first look at the concepts that are fundamental to information security.

2.2.1 Security Concepts

2.2.1.1 Confidentiality

Confidentiality deals with the protection of sensitive information. This is typically influenced by rules stating the disclosure restrictions of the protected information. Making data incomprehensible is the most common form of confidentiality. This is achieved by using encryption.

The two types of encryption techniques which exist include conventional encryption (based on symmetric encryption algorithms) and public-key encryption (based on asymmetric encryption algorithms). Both types are discussed in the following subsections.

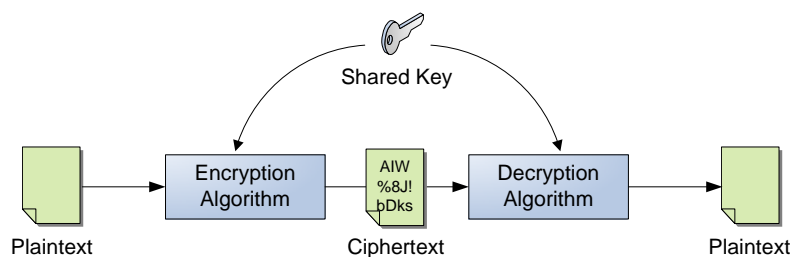


Figure 2.4: The basic model for conventional encryption.

Conventional Encryption Conventional encryption techniques were, for a long time, the only known form of encryption. In fact, these are still the most widely used type today. Symmetric encryption relies on the existence of a *shared secret* between the message source and the destination.

Figure 2.4 illustrates how conventional encryption works. The *plaintext* (i.e. the message to be protected) and the *shared key* (i.e. the shared secret) are passed into the encryption algorithm to produce the *ciphertext* (i.e. the message encrypted). The sender sends the ciphertext to the recipient. Once received, the recipient passes the ciphertext and the shared key into the decryption algorithm to reveal the plaintext.

When using symmetric encryption, the ability to derive the plaintext from the ciphertext alone must be impractical. Additionally, the security of symmetric encryption techniques must rely on the secrecy of the key – not the encryption algorithm. Knowledge of the encryption algorithm should not provide a security risk.

Widely accepted symmetric encryption algorithm standards today are Data Encryption Standard (DES), Triple-DES and Advanced Encryption Standard (AES). An overview of these standards can be found in [71]. DES operates by splitting data into 64-bit blocks and encrypting them using a 56-bit cipher key. However, concerns exist over this standard. Many believe that the short key size makes it vulnerable to brute-force attacks. Furthermore, the internal structure of DES is not completely open raising concerns that backdoors may exist which enable the deciphering of messages without the key. Triple-DES was proposed to alleviate the fears of brute-force attacks against DES. With Triple-DES, two 56-bit keys are used in a three step process. Firstly, the plaintext is encrypted with key 1, then the result is decrypted with key 2, before finally, the result of this is encrypted with key 1. This process changes the cost of discovering the encryption keys using a brute-force attack from the order of 2^{56} to

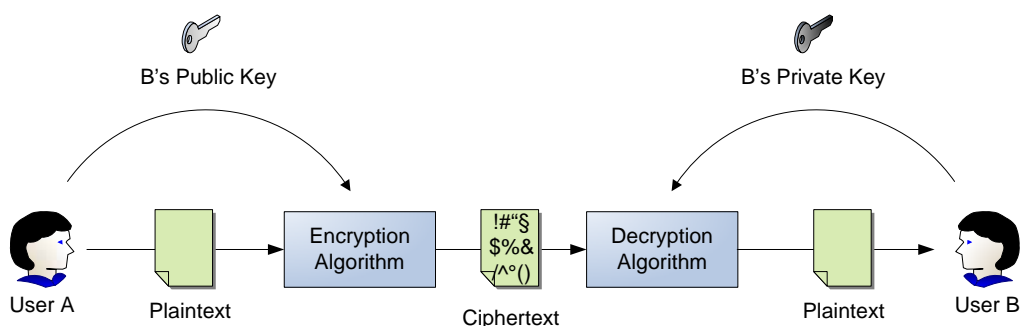


Figure 2.5: The basic model for public-key encryption.

2^{112} .

Although Triple-DES addresses the problem of the small key size, it performs relatively slowly when implemented in software and it is not suitable for use in limited-resource platforms. To resolve these problems, Rijndael was proposed in 1997 and this algorithm was later officially standardized as AES in 2001. AES operates fast in both hardware and software, requires little memory to operate, uses 128-bit block sizes, and supports key lengths of 128, 192 and 256 bits. Additionally, the specifications of AES are open and can be found in [26].

Public-key Encryption A difficult issue facing conventional encryption methods is key distribution. These methods require either (1) the communicating parties to already possess the secret key or (2) a key distribution centre (KDC) to be in place. But with (2), maintaining total secrecy over the communications is under threat. Key owners are forced to share keys with a KDC which could potentially become compromised.

Public-key encryption schemes tackle this issue by using two separate keys rather than a single secret key. Public-key (or asymmetric) encryption algorithms operate by using one key for encryption and the other key for decryption. An important characteristic of asymmetric encryption algorithms is that it must be computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key [70].

Figure 2.5 shows how public-key encryption works. In this example, user B owns a key pair – a public key (to be distributed to everyone) and a private key (to be kept secret). Suppose user A decides to send an encrypted message to B. The message is encrypted by A using B's public key

and the resulting ciphertext is sent to B. When received, B decrypts the ciphertext using his private key to reveal the plaintext. Total secrecy over the communications is achieved as private keys can be generated locally and never need to be distributed to anyone.

The most widely accepted asymmetric encryption algorithm in existence today is the Rivest-Shamir-Adleman (RSA) scheme [56]. RSA is a block cipher and uses mathematical functions (rather than substitution and permutations) to perform encryption/decryption functionalities. A drawback of asymmetric encryption techniques is that they are computationally slower than symmetric encryption techniques. For this reason, it is common practice for a secure communication session to initially use asymmetric encryption for key negotiation before using symmetric encryption once a secret key has been established.

2.2.1.2 Integrity

Integrity is concerned with protecting messages from unauthorized modifications. To achieve this, digital signatures can be used. Creating a digital signature involves taking a fingerprint of the message (otherwise known as a *hash value*) and then encrypting this fingerprint with the signer's private key. Digital signatures make use of public-key encryption.

To create the fingerprint, hash functions are used. Hash functions take the form $h = H(M)$ where h represents a fixed-length hash value, H represents the hash function and M represents a variable-length message. Some important requirements of hash functions include [70]:

1. be able to be applied to variable-length data;
2. be able to produce a fixed-length output;
3. be relatively easy to compute for any given message;
4. be computationally infeasible to derive the message from the hash value (*one-way* property);
5. be computationally infeasible to find another message which produces the same hash value;
6. be computationally infeasible to find for any pair (x, y) , such that $H(x) = H(y)$.

Hash functions satisfying the first five requirements are known as *weak hash functions*. For those which also satisfy the sixth requirement (which protects against the birthday attack [72]), these are known as *strong hash functions*.

A simple example of a hash function is one which performs a bit-by-bit exclusive-or (XOR) operation over each block of the message. Alternatively, this simple example could be enhanced by incorporating a one-bit circular shift (rotation) on the hash value after each data block is processed. More elaborate hash function algorithms include the MD5 message-digest algorithm [55] and the Secure Hash Algorithm (SHA) [25].

To check if message integrity has been maintained, digital signatures need to be verified at the receiver's end. This verification involves a two-step process. Firstly, the receiver decrypts the signature using the signer's public key to reveal the fingerprint of the message. Secondly, the receiver performs its own hash over the message to create a fingerprint and if the two fingerprints match, integrity has been maintained.

2.2.1.3 Authentication

Authentication addresses the question of: "Are you who you say you are?". This principle is concerned with verifying an entity's identity claim on the basis of some kind of evidence that no-one else is able to present. Authentication can be based on three types of evidence [74]:

1. Something the user knows.
2. Something the user has.
3. Something the user is or does.

Passwords are a classic example of authentication based on something that the user knows. Token devices (e.g. memory cards and smart cards) are a common form of authentication based on something that the user has, whereas authentication based on biometrics (e.g. fingerprints and retina scans) fall under the category of something the user is or does.

The most basic type of authentication is one-way authentication – otherwise known as *simple authentication*. This involves a claimant who wishes to authenticate itself with a verifier by sending a

proof of identity. A more advanced form of authentication is *mutual authentication*. This form of authentication enables two-way authentication so that the claimant can be assured that it is authenticating to the correct verifier.

2.2.1.4 Authorisation

Authorisation refers to the process of granting or denying access to resources. It is closely associated to access control which defines the models and mechanisms for applying restricted access to protected resources. The most common access control models typically consist of three main components:

- Subjects – the entities (e.g. users) of the system.
- Objects – the resources (e.g. files) under protection.
- Permissions – the access rights supported by the model. Common access rights include read, write, execute and own.

An important aspect of authorisation is that it assumes that subjects have been properly and successfully authenticated beforehand. This enables authorisation systems to enforce their policies and to make decisions based on the identities of subjects.

2.2.1.5 Non-repudiation

Non-repudiation is concerned with providing guarantees so that, for example, a sender of a message cannot deny that a message transmission never occurred when, in fact, it did. Similarly, it can be used to prevent a recipient from claiming that it never received a transmitted message when it was actually successfully received. Dispute resolution is a major reason for non-repudiation.

According to [21], digital signatures are the only means of achieving non-repudiation without involving a notary. Following are the different forms in which non-repudiation can take [21]:

- Non-repudiation of creator – prevents the creator from denying ever creating the data.
- Non-repudiation of sender – prevents the sender from denying ever sending the data.

- Non-repudiation of submission – proof of data transmission for the sender. Prevents the communication system or any recipients from claiming that the data was never transmitted.
- Non-repudiation of delivery – prevents the recipient from denying ever receiving the data. It proves to the sender that the data was delivered to the intended recipient.
- Non-repudiation of receipt – prevents the recipient from claiming that he has never seen the data.
- Non-repudiation of use of service – protects a service provider from a user denying that he ever used a particular service offered by the provider.

2.2.1.6 Auditing

Auditing refers to the logging of system events and keeping an audit trail. It is concerned with keeping users accountable for their actions. An important characteristic of auditing is that it enhances access control. In principle, if unauthorised actions pass the access control system, then the auditing system detects it. Auditing enhances access control in the following ways [60]:

- It acts as a deterrent. Users become more reluctant to perform unauthorised activities if they know that their actions are being tracked.
- Log files provide evidence for investigating attempted or actual security violations. It helps with discovering security holes in the system.
- Ensures that authorised users do not abuse or misuse their privileges.

2.2.1.7 Trust

When examining the issue of trust, it is important to identify who trusts whom on what. The principle of trust suggests a relationship between (at least) two entities based on a particular action. Trust can be categorised into two types – direct trust and third-party trust. The former type describes the situation where two entities have established a trust relationship themselves. The latter refers to an implicit trust relationship between two entities since both of these entities trust a specific trusted third-party

(the third-party determines the trustworthiness of the two entities). Third-party trust is common in public-key cryptography where a Certification Authority (CA) acts as the trusted third party.

2.2.2 Security Policies and Models

A security policy is simply a statement of what is allowed and what is not allowed to occur between entities in a system.

We focus on authorisation policies – a type of *prevention* policy. Other policy types include *detection* policies and *recovery* policies.

Authorisation policies can be expressed mathematically, specifying the allowed and disallowed states. Let us consider the following, where the computer system is expressed as a finite-state automaton with a set of transition functions that change state [6]:

Let P be the system state space.

$Q \subseteq P$ defines the states in which the system is authorised to exist in.

When the system is in a state $s \in Q$, the system is secure.

When $s \in P \setminus Q$, the system is not secure.

An authorisation policy characterises Q . In other words, the policy distinguishes between authorised (or secure) system states and unauthorised (or non-secure) system states. A system mechanism (e.g. access control) prevents the system from entering $P \setminus Q$. A secure system is a system that starts in an authorised state and cannot enter an unauthorised state.

The remainder of this section discusses several policy models, starting with the Chinese Wall policy model.

2.2.2.1 Chinese Wall Policies

The *Chinese Wall* security policy was proposed by Brewer and Nash [7] to address *conflict of interest* issues related to consulting activities within the commercial environment.

For example, the policy model can be viewed as a “code of practice” to prevent consultants from disclosing sensitive information (or *insider knowledge*) of one corporation to a competitor. Consultants can, however, freely advise corporations which are not in competition with each other, and also

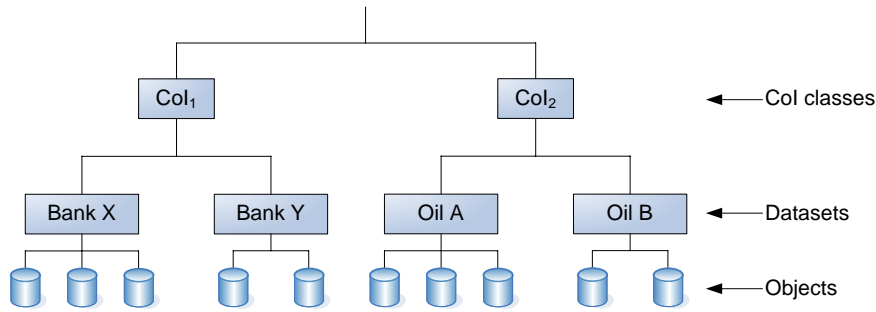


Figure 2.6: Chinese Wall data hierarchy.

access general market information.

A Chinese Wall policy is a form of MAC (Section 2.2.3.2). It aims to prevent the occurrence of information flows that could cause a conflict of interest. Initially, the user is free to access any object. But after the initial choice is made, a Chinese Wall is created for that user around the dataset containing the selected object.

The model organises all data into a hierarchy:

1. At the lowest level, there are individual items of data, concerned with a single corporation. Such data is stored as *objects*.
2. At the intermediate level, objects are grouped according to their respective corporation. Each group of objects is referred to as a *dataset*.
3. At the highest level, the datasets of those corporations in competition with one another are grouped. Each group of datasets is referred to as a *conflict of interest (CoI) class*. Each dataset belongs to only one CoI and each CoI has two or more member datasets. Hence CoI classes are mutually disjoint.

Let us consider the example illustrated in Figure 2.6 which has been adapted from [7]. If *Alice* has read sensitive information of *Bank X*, then *Alice* is prohibited from reading sensitive information from any other dataset (i.e. bank) within CoI_1 . In other words, read access is only granted if the object requested (a) is in the same dataset of an object previously accessed by that user, or (b) belongs to a CoI class not previously accessed by that user.

But suppose there are two users – *Alice* and *Bob* – where *Alice* can access *Bank X* and *Oil A*, and *Bob* can access *Bank X* and *Oil B*. If *Alice* reads sensitive information from *Oil A* and writes it to *Bank X*, then *Bob* can read sensitive information of *Oil A*. For this reason, the *write* rule is very restrictive. A user who has previously read objects from more than one dataset cannot write any object. As a result, the flow of information is confined to its own company dataset.

The model also considers sanitized information (i.e. where information is disguised so that the corporation’s identity is not discovered). For such information, access restrictions are lifted.

2.2.2.2 Bell-LaPadula Model

The Bell-LaPadula (BLP) Model [3] uses a hierarchical set of *security classes* to impose access restrictions. It was initially designed for use in military environments. Rather than placing constraints on the inter-relationships between objects (as is the case with Chinese Wall policies), each object within the BLP model is labelled.

To explain how this model works, let us consider that there exists the security classes Top Secret (*TS*), Secret (*S*), Confidential (*C*) and Unclassified (*U*), such that $U \leq C \leq S \leq TS$.

Objects are assigned to these security classes to indicate their *sensitivity* level. An object’s security label has the form (*class*, {*cat*}) where *class* is the security class and *cat* is the category of the information.

Subjects are also assigned to these security classes to indicate their *clearance* level. A subject’s security attributes have the form (*clear*, {*NTK*}) where *clear* is the maximum security class to which the subject is permitted access and *NTK* is the subject’s need-to-know (representing the categories to which the subject is permitted access).

Figure 2.7 illustrates these security classes, showing subjects and objects allocated to these classes.

Access to an object by a subject depends on the security levels associated with the two. In particular, the following principles must hold:

read down – a subject’s clearance must dominate the security level of the object being read. In other words, subject *S* is allowed to read object *O* only if $\text{class}(O) \leq \text{class}(S)$. This is known as the *simple security* rule. The subject’s need-to-know must also include the object’s category.

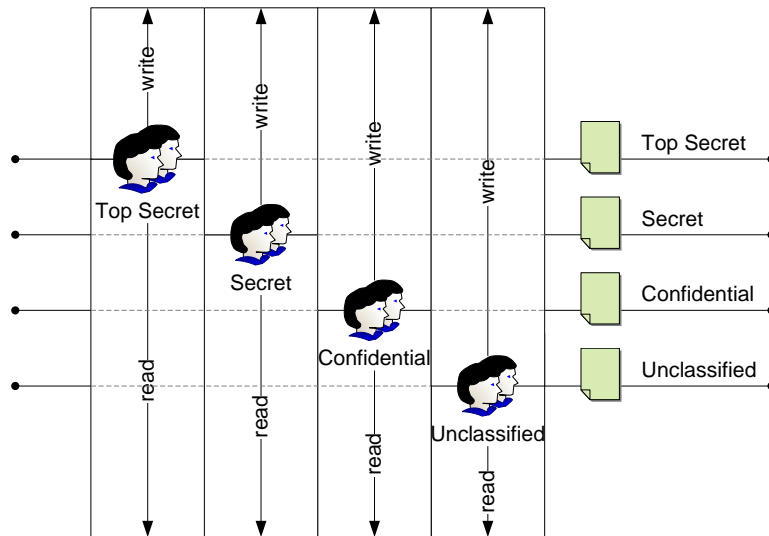


Figure 2.7: Bell-LaPadula model example.

write up – a subject’s clearance must be dominated by the security level of the object being written.

In other words, subject S is allowed to write object O only if $\text{class}(S) \leq \text{class}(O)$. This is known as the **-property* rule. The subject’s need-to-know must also include the object’s category.

Compliance with these principles prevents information in high-level resources (i.e. more sensitive) to flow to resources at lower levels. This is illustrated in Figure 2.7. In this model, information can only flow upwards or within the same security class.

Although the BLP model is well-suited for upholding confidentiality, it suffers from several limitations [6]:

1. The model is static. Access rights and security classes cannot be changed.
2. How to create and delete subjects and objects is not specified. This limitation can be addressed using the Clark-Wilson model [11].
3. The model does not deal with integrity, only confidentiality. This limitation can be addressed using the Biba model [5] (and also the Clark-Wilson model [11]).
4. It contains covert channels (i.e. information flows that violate the security policy).

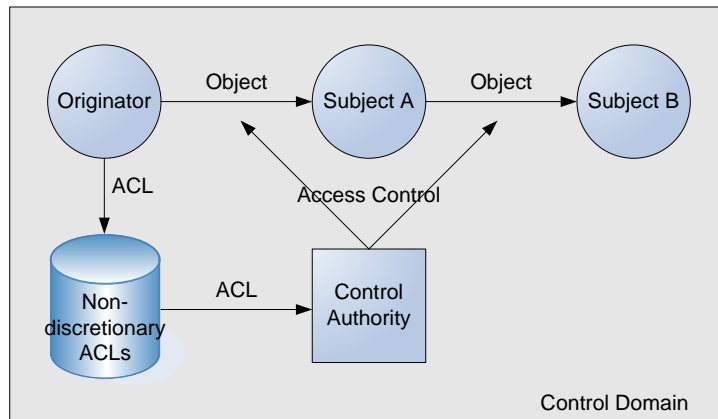


Figure 2.8: ORCON example.

2.2.2.3 ORCON

Originator Controlled (ORCON) [32] is concerned with controlling the dissemination of objects. This policy model requires recipients to obtain approval from the object’s originator in order to distribute the object to other subjects.

The term *originator* refers to the subject responsible for the data contained in the object and for determining to whom the data can be released. This differs from the term *owner*, who is the subject responsible for the creation of the object and is authorised to change the Discretionary Access Control (DAC) permissions on the object [32].

Traditional solutions focus on enforcing ORCON policies within a closed control environment. In these cases, policies are typically centrally controlled and users behave within the scope of the policies. Figure 2.8 illustrates the design of a traditional ORCON solution [52].

In this diagram, the originator releases the object (marked with “ORCON”) and makes it available to subject *A* in the policy settings (ACL). Should *A* decide to pass the object to subject *B*, the control authority must first check whether *B* is permitted access to the object. Consequently this enforcement allows the originator to maintain control over how the object is distributed.

More recently, Usage Control (UCON) [52, 53] has emerged as a means of controlling and managing the usage of objects. When coupled with UCON, ORCON policies can be enforced beyond the originator’s local control domain. This provides a solution for controlling dissemination and re-dissemination beyond a closed control environment where a central authority is not available.

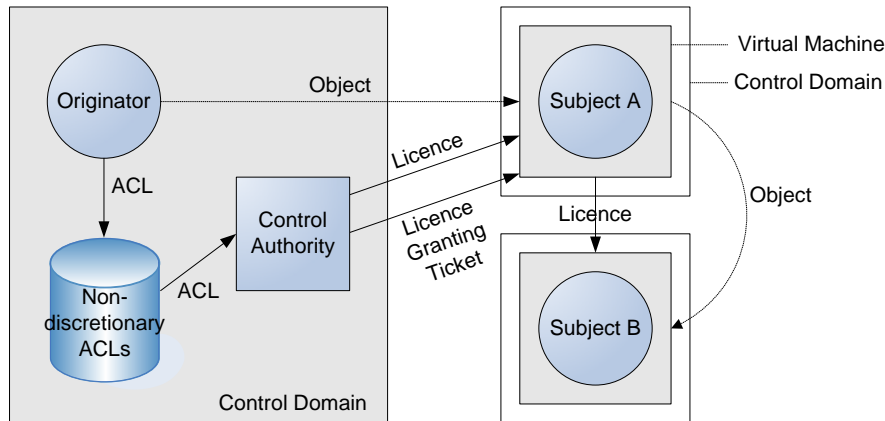


Figure 2.9: ORCON with UCON example.

The UCON model defines several components. The object is encapsulated into a cryptographically protected *digital container*. The encapsulated object can only be accessed from special application software or hardware (known as a *virtual machine*). Access rights are listed in a *control set* (implemented as licences for propagating usage rights to authorised users). A licence is required for the user’s virtual machine to access the encapsulated object. Finally, a *control center* exists for storing security policies and managing access rights.

Figure 2.9 shows how ORCON and UCON can be combined to enforce access control policies beyond the originator’s local control domain. Subject A and B reside in separate control domains. Both subjects require a licence to access the object. In this particular example, subject A is issued a licence directly from the originator (via the control center) and also a *licence granting ticket*. This ticket allows A to issue licences to other authorised subjects (in this case, to subject B). The shaded areas (i.e. the originator’s control domain and the virtual machines) indicate the extent of the originator’s control over the disseminated object.

2.2.2.4 Digital Rights Management

Similarly to ORCON, Digital Rights Management (DRM) is also concerned with controlling the distribution and usage of objects. DRM is highly driven by media companies, who are increasingly striving to protect commercial digital content and combat digital piracy. These companies argue the need for DRM because “without protection and management of digital rights, digital content can be

easily copied, altered and distributed to a large number of recipients, which could cause revenue loss (to them)” [44].

DRM systems protect high-value digital assets by protecting against unauthorised access to the content, controlling the distribution of these assets and managing the content usage rights. Central to achieving this is the use of *digital licences* that grant certain usage rights to the consumer. Digital content is encrypted before being distributed and licences are purchased by the consumer to unscramble and access the content. Client-side applications (e.g. media players) are critical in DRM implementations in that they enforce protection based on these licences.

Many DRM models and implementations have been proposed for various industries (e.g. online music, film, print, etc.) and different platforms (e.g. personal computers, mobile phones, embedded systems, etc.) [24, 42, 45]. Despite this, the basic DRM process is the same and usually involves four entities: the *content provider*, the *distributor*, the *clearing house* and the *consumer* [44]. A typical DRM system is shown in Figure 2.10 [44]. Each component in this system is described as follows:

Content Provider – Owns the intellectual property rights to the digital content and protects these rights.

Distributor – Receives digital content from the content provider and distributes this content to consumers. In other words, acts as the distribution channel.

Consumer – Receives digital content from the distributor and purchases licences from the clearing house to access the content.

Clearing House – Issues digital licences and handles payments. That is, the clearing house receives payments from the consumer, pays royalty fees to the content provider and pays distribution fees to the distributor.

There is much debate over the usefulness of DRM [24]. Some of the main arguments against DRM are:

- No DRM standards exist. Most DRM models and technologies are proprietary.
- Restricts innovation, research, free speech and public access to digital information.

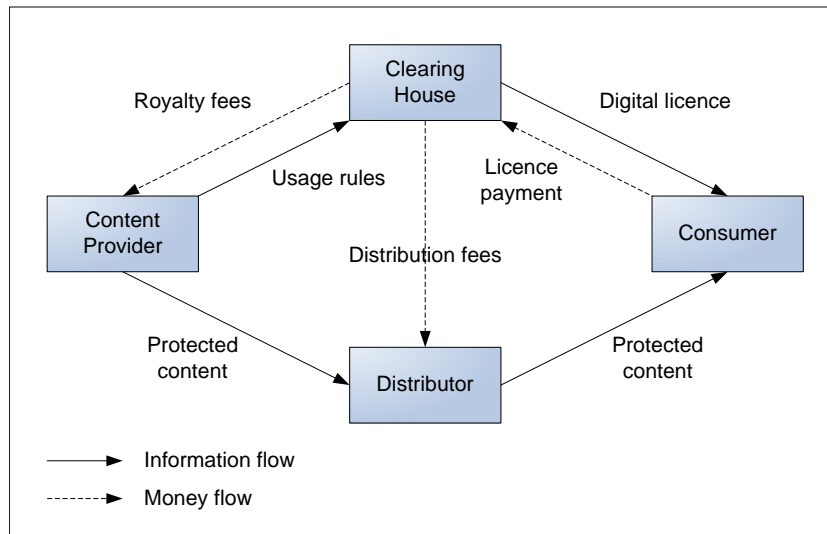


Figure 2.10: A typical DRM system.

- Risk of data and privacy infringement.

Despite these arguments, DRM provides the following benefits:

- Allow consumers to acquire and use digital content legally.
- Allow intellectual property rights to be protected.
- Assure the authenticity and integrity of the digital content.

2.2.2.5 Information Flow Models

We have already seen examples of information flow models in the previously discussed security models – namely, Chinese Wall policies (Section 2.2.2.1) and the Bell-LaPadula model (Section 2.2.2.2). Another security model that constrains information flow is the Biba model [5].

Although similar to the Bell-LaPadula model, the Biba model differs in that it focuses on *integrity* and not confidentiality. This model contains *integrity levels* which are analogous to the sensitivity levels used in the Bell-LaPadula model (e.g. Top Secret, Confidential, etc.). Data integrity is maintained by conforming to the following principles:

read up – a subject’s clearance must be dominated by the integrity level of the object being read.

This is known as the *simple integrity property*.

write down – a subject’s clearance must dominate the integrity level of the object being written. This is known as the *integrity *-property*.

These principles ensure that modifications can only flow downwards or within the same integrity level, hence upholding the integrity of information in the more sensitive, higher level resources. They are the opposite to the principles defined in the Bell-LaPadula model. As a result, a system cannot use both of these models (otherwise there would be no information flow at all).

Some recent examples of information flow models are discussed in [64, 67]. Shen and Qing [64] define an information flow system as a state machine model and view information flow as a sequence of meta-flows (representing the transfer of information from variable v_1 to v_2 by subject s executing primitive (atomic task) p in system-state ss). Information flow constraints are defined in terms of *static* security policies (which do not vary with system states) and *dynamic* security policies (which vary with system states). This model provides general definitions for expressing information flow properties, as well as expressing security policies and access control in terms of information flow. This is in contrast to the previous models which specify precise rules and constraints for securing information flow.

Other types of information flow models are more specialised. For example, Singaravelu *et al.* [67] define an information flow model for protecting sensitive information in service-oriented architectures using web services. This model separates the underlying implementation architecture into two parts: (1) a small, functionally-limited, trusted platform (for handling sensitive information) and (2) a large, functionally-rich, untrusted platform (that invokes the other platform when sensitive information needs to be handled). This intention behind this strategy is to ensure that access to the sensitive information is limited to only the web services components that are required to process this information. Restricting sensitive information to the trusted platform minimises the potential for this information being exposed as a result of attackers exploiting security vulnerabilities in the system.

2.2.3 Traditional Access Control Models

Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role-Based Access Control (RBAC) all represent traditional approaches towards access control enforcement. We now elaborate on each of these models.

2.2.3.1 Discretionary Access Control

The DAC model adopts an *individual resource ownership* approach [27]. It is *owner-controlled* in that object owners control the access permissions applied to their objects. To a large extent, authorisation support in many mainstream systems is based on DAC (including UNIX operating systems, Windows operating systems and Apache Web servers).

The discretionary model functions on the basis of user identities and authorisation policies. Permissions on objects are assigned to subjects (the authorisation policies specify the permissions that each subject is granted on each object). The advantage of this model is that it is highly flexible. However, due to issues based on delegation of rights and uncontrolled dissemination [60], the discretionary model is considered to be less suitable for corporate environments (where organisational structures exist).

Access Control Lists (ACL), which specify the access rights of each subject in the system for a particular object, are commonly used for implementing the discretionary model. Each object possesses a separate ACL. With ACLs, it is easy to (a) determine the access rights of each subject on an object and (b) revoke all access rights on an object. The drawback for ACLs is that for any given subject, it is difficult to determine all of their access rights on all objects in the system. Revoking each and every access right for a particular subject on all objects is also difficult.

Capability lists are another DAC implementation option. Where ACLs are allocated to objects, capability lists are allocated to subjects. In other words, each subject possesses a separate capability list. It lists the objects, together with the subject's access permissions for each object.

Figure 2.11 shows an example ACL (top row) and capability list (bottom row). The ACL shows the access permissions assigned to the object *File 1* whereas the capability list shows the access permissions granted to the subject *Alice*.

2.2.3.2 Mandatory Access Control

The MAC model adopts an *organisational resource ownership* approach [27]. It is *organisation-controlled* in that information flows are restricted. This type of security is also known as multi-level security (MLS). Many MLS systems are based on the Bell-LaPadula model (discussed in Section 2.2.2.2).

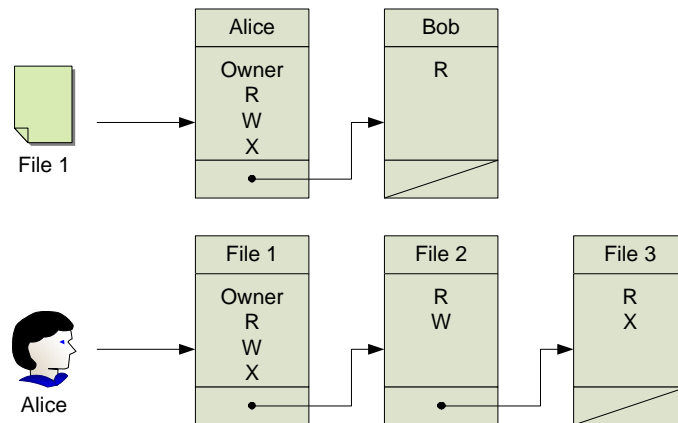


Figure 2.11: ACL and capability list examples.

MAC enforces access control by classifying all subjects and objects in the system. Hence it is based on a security label system. Each subject is assigned a *security clearance* (which reflects the trustworthiness of the user not to disclose sensitive information to unauthorised users). Objects are similarly classified to reflect the sensitivity of its content. A subject is granted access if their clearance is equal to or higher than the object's classification.

An example of a set of classifications (ordered from highest to lowest) is: *top secret*, *secret*, *confidential* and *unclassified*. As well as the classification, security labels include different *categories* (representing compartments of information within the system). Unlike classifications, categories do not follow a hierarchical scheme. The purpose of categories is to enforce need-to-know rules. For example, having a top secret security clearance should not permit the subject access to all top secret information. Categories may be based on departments, such as *marketing*, *sales*, *human resources*, etc. Suppose user Alice has the security clearance $\{\textit{secret}, \{\textit{marketing}, \textit{sales}\}\}$, then access to a file labelled as $\{\textit{confidential}, \{\textit{human resources}\}\}$ would be denied.

This model is considered most suitable for rigid environments (such as the military) where information is owned by a central authority and where confidentiality is of utmost importance. Apart from being too inflexible for many environments, this model also has the drawback that it cannot properly represent user roles.

2.2.3.3 Role-Based Access Control

Role-Based Access Control (RBAC) [22, 23, 29, 59] attempts to overcome the drawbacks associated with the discretionary and mandatory models. In a shift from mapping the permissions of individual subjects directly to objects, RBAC introduces the concept of *roles* (which represent job functions) and operates by granting permissions to roles before assigning users to these roles. Roles can reflect specific task competencies, responsibilities or even specific duties [59].

Key motivating factors behind RBAC's development include (1) simplifying management procedures and (2) the need for specifying and enforcing enterprise-specific policies. The introduction of this model has produced the following advantages:

Ease of administration – allocating users to roles typically requires much less technical expertise than determining suitable access rights for each role. As user assignment to roles changes much more frequently than access permission mappings to roles (which is considered to be relatively stable), this greatly simplifies administration of the system once the RBAC framework has been established.

Increased flexibility – apart from roles, configuration of the RBAC framework can also include role hierarchies, relationships and constraints. Additionally, policy specifications can be based on what actions, when, from where, in what order, and in some cases, under what relational circumstances [22].

Delegation of administration – for example, global-level policies affecting the whole domain can be defined and administered centrally at the enterprise level whereas policies relating to a specific sub-domain can be enforced locally at the organisational unit level.

RBAC (specifically RBAC96 [58]) consists of four conceptual models:

- RBAC₀ contains the minimal features of a system implementing RBAC.
- RBAC₁ adds support for role hierarchies.
- RBAC₂ adds support for constraints (e.g. separation of duty).
- RBAC₃ includes all aspects of the above models.

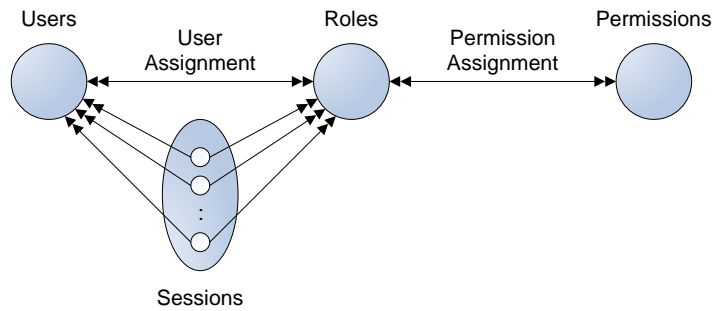


Figure 2.12: The Role-Based Access Control model (Core RBAC).

Figure 2.12 illustrates the simplest form, $RBAC_0$ (otherwise known as Core RBAC). A user can be mapped to multiple roles and a role can have multiple users mapped to it. Similarly, a permission can be mapped to multiple roles and a role can contain multiple permissions. There is also the notion of sessions. A session represents an instance of a user connected to the system and defines the subset of activated roles. Different sessions for the same user can be active.

RBAC suits multi-user, multi-application systems and promotes two important features – *least privilege* and *separation of duty*.

Least privilege describes the administrative practice of selectively assigning permissions to users. The objective is that the user should not be given more permissions than necessary to perform their job function.

Separation of duty (SoD) [66] is a form of constraint, designed to prevent users from exceeding a reasonable level of authority and to prevent failures caused as a result of collusion amongst individuals. The two main categories are *static SoD* and *dynamic SoD* [27]:

- Static SoD – places constraints on the assignment of roles to avoid conflicts of interest.
- Dynamic SoD – similar to static separation but based on time or other dynamic factors.

2.2.4 Implementing Access Control

Access control implementations typically include a *reference monitor* [2] – a layer of functionality that safeguards access to protected objects. Reference monitors play the crucial role of validating access requests (i.e. to verify a subject’s right to access a protected object every time the object is

referenced).

The primary building blocks of access control implementations are [27]:

- Policy Enforcement Points (PEP) – intercept access requests, send authorisation decision requests to PDPs and enforce authorisation decisions. The PEP is representative of a reference monitor.
- Policy Decision Points (PDP) – accept authorisation decision requests from PEPs and make authorisation decisions based on the authorisation policies provided by PMAs.
- Policy Management Authorities (PMA) – create, supply and maintain authorisation policies.

Access control implementations can be compared based on the level of policy expressiveness that they support. Common forms of authorisation include [27]:

- Identity-centric authorisation – based on subject and object identifiers. This is the simplest form of authorisation. For example, *Alice* may access file *X*.
- Category-aware authorisation – extends on identity-centric by supporting structured sets of subjects and objects (independent of one another). This involves subject attributes (e.g. group memberships, role assignments) and object attributes (e.g. object classifications, object properties, etc.). For example, *Developers* may access files within directory *dev*.
- Relationship-aware authorisation – extends on category-aware by supporting jointly structured subjects and objects. This provides the ability to express conditional relationships and to evaluate them at runtime. For example, *Developers* may access files within directory *dev/region3* if the user is based in the same region.

2.2.5 Emerging Access Control Technologies

A limitation with the traditional access control models is that they do not fully satisfy the requirements for distributed access management (needed for inter-organisational scenarios). This is because these models mainly focus on intra-organisational environments. They implicitly assume that identity and resource providers reside in the one organisation.

Several technologies are now emerging that focus on *federation*. In other words, they aim to allow organisations to extend their existing systems to accommodate collaboration with partner organisations. This section looks at some of these emerging access control technologies.

2.2.5.1 Security Assertion Markup Language

The Security Assertion Markup Language (SAML) is an XML-based framework for exchanging security-related information. Developed by an OASIS [51] technical committee, the current specification release (at the time of writing) is version 2.0 [62] (published as an OASIS standard). This release improves on SAML version 1.3 [61] and includes Liberty Alliance [43] contributions.

This framework plays a significant role in providing federated identity management systems and integration amongst heterogeneous environments. It is designed for cross-domain authorisation services and single sign-on.

Of all the components in the SAML specification suite, assertions are fundamental. An assertion is simply a statement (or “declaration of fact”) based on a subject and is issued by an issuing authority. There are three types of assertions in SAML:

Authentication assertions – An issuing authority asserts that subject S was authenticated by means M at time T . These assertions are essentially proof of a past authentication event and are targeted towards single sign-on.

Attribute assertions – An issuing authority asserts that subject S is associated with attributes A , B , etc. with values “a”, “b”, etc. Aimed towards authorisation services and distributed transactions, these assertions typically contain information contained in directories (e.g. LDAP repositories).

Authorisation Decision assertions – An issuing authority decides whether to grant the request by subject S for access type A to resource R given evidence E . Again, these assertions are used by authorisation services and distributed transactions.

Additional components of the SAML specification suite include the SAML protocols, bindings and profiles. SAML protocols define the process of how assertions are requested (e.g. the request and response messages involved). Bindings refer to how the SAML protocol messages are trans-

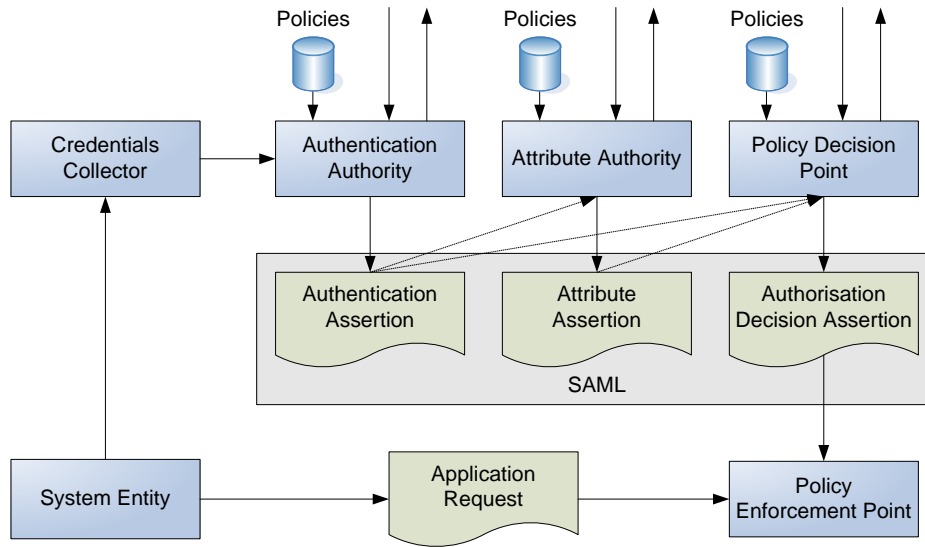


Figure 2.13: The SAML producer-consumer model.

mitted across lower-level application protocols such as HTTP and SOAP. Profiles define how SAML protocols and bindings (as well as assertions) can be combined to solve specific business problems.

The main properties of SAML include [27]:

- SAML authentication assertions do not constrain the initial authentication method. They simply contain information on the method as well as metadata on the initial authentication context (e.g. length of authentication keys).
- SAML does not specify attribute vocabularies. It is generic in terms of the attributes that it can carry.
- SAML assertions can be passed to multiple relying parties. They do not need to be scoped to dedicated targets.
- SAML relies on XML Signature [93] and XML Encryption [91] for object protection.

Figure 2.13 displays the SAML producer-consumer model. This model illustrates the flow of events of how SAML assertions are produced and consumed.

The Credentials Collector collates the System Entity’s credentials and passes them to the Authentication Authority. In this model, the System Entity is the subject. The Authentication Authority

assesses these credentials in accordance to its policies and then issues an Authentication Assertion. From here, one of two paths can be followed. The Authentication Assertion can be passed directly to the Policy Decision Point (PDP) or the assertion can be passed to the Attribute Authority. As before, the Attribute Authority receives the Authentication Assertion, assesses it according to policy and issues an Attribute Assertion which can then be passed onto the PDP. When the PDP receives these assertions, it also assesses them according to its own policies and issues an Authorisation Decision Assertion which is then forwarded to the Policy Enforcement Point (PEP). Meanwhile, the System Entity has sent a request to the PEP to access a particular service. Using the information in the Authorisation Decision Assertion, the PEP can either permit or deny the request.

2.2.5.2 eXtensible Access Control Markup Language

The eXtensible Access Control Markup Language (XACML) is an XML-based language for access control. Similarly to SAML, this technology is also developed by an OASIS [51] technical committee. The current specification release (at the time of writing) is version 2.0 [20] (published as an OASIS standard).

The objectives behind XACML are (1) forming a standardised approach for access control and (2) the ability to provide finer granularity for making access control decisions. XACML defines:

- A syntax to express authorisation policies (in other words, an authorisation policy language).
- A syntax to express authorisation decision requests and responses.

XACML can make access control decisions based on *dynamic* information. Additionally, the XACML specification describes an architecture for access control enforcement. This architecture is illustrated in Figure 2.14.

An authorisation request sent by the subject (access requester) is received by the Policy Enforcement Point (PEP). The PEP is responsible for accepting requests and for enforcing access control decisions. To determine how to handle this request, the PEP creates an XACML request and sends this to the Policy Decision Point (PDP) so that a decision can be made regarding whether to permit or deny the requested action. The PDP retrieves the applicable policies from the Policy Access Point (PAP) – the data store for the access control policies. Additionally, the PDP may also request attributes

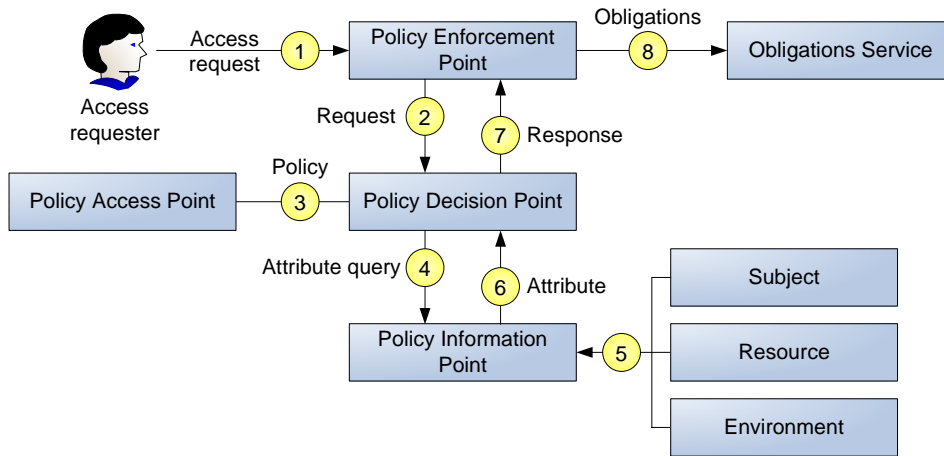


Figure 2.14: The XACML architecture.

from the Policy Information Point (PIP) so that the policies can be evaluated. The types of attributes provided by the PIP include subject attributes, resource attributes and environment attributes. This component provides XACML with access to dynamic statistics for making access control decisions. Once the applicable policies have been evaluated, the PDP sends its authorisation decision to the PEP who then enforces this decision. The PDP can also state obligations that need to be fulfilled by the PEP before access is granted.

Policies in XACML consist of several components: target, rules, rule-combining algorithm and obligations.

Target – The target determines the applicability of the policy for the authorisation request. Determining this is achieved by comparing the attributes of the target (i.e. subject, resource, action and environment) against the equivalent attributes in the request message. Should these values match, the policy is then declared to be relevant for the request. A policy contains only one target.

Rules – Policies can contain multiple rules and each rule consists of the following elements: a *condition*, an *effect* and a *target*. When the condition is evaluated, it returns either *true*, *false* or *indeterminate*. If the condition returns true, the rule returns the value of the effect element (i.e. *permit* or *deny*). If the condition returns false, the rule returns *not applicable* and if the condition returns indeterminate, the rule also returns *indeterminate*. The target element of a rule is

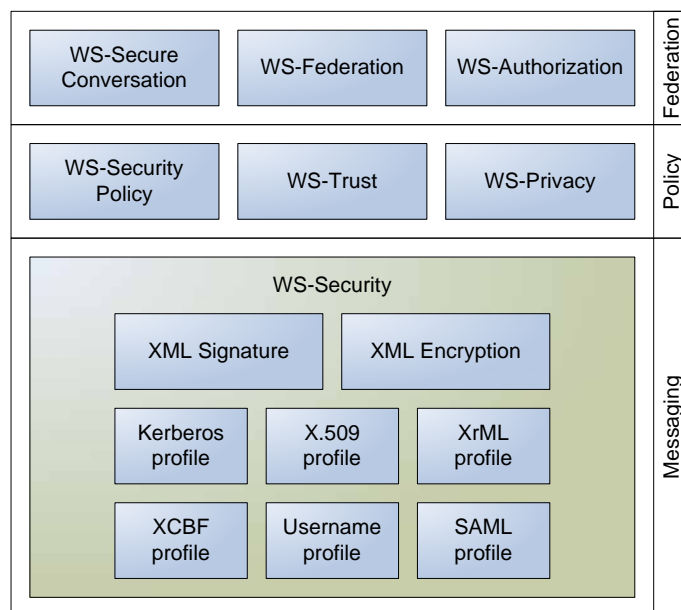


Figure 2.15: The Web Services Security framework.

comparable to the target element of a policy – i.e. is the rule relevant for the request?

Rule-combining algorithms – As policies can contain multiple rules, the rule-combining algorithm is responsible for resolving conflicting rule results so that a single authorisation decision can be arrived at. Examples of these algorithms include *deny-overrides*, *ordered-deny-overrides*, *permit-overrides*, *ordered-permit-overrides* and *first-applicable*.

Obligations – Obligations are actions which must be fulfilled by the PEP in conjunction with executing the authorisation decision. This provides XACML with finer granularity access control.

2.2.5.3 Web Services Security

The WS-Security roadmap [63] outlines a number of emerging specifications focusing on identity and access management for Web services (WS) technologies. A partial overview of the Web services security framework is shown in Figure 2.15.

The WS-Security [83] specification focuses on message integrity and message confidentiality. More precisely, it describes how signatures and encryption headers are attached to SOAP [68] messages. Additionally, it defines various *profiles* for attaching security credentials to messages (including

binary security tokens such as X.509 certificates and Kerberos tickets).

The level above WS-Security is the Web services policy layer. This layer consists of the following specifications:

WS-SecurityPolicy [88] – for expressing security-specific requirements and capabilities in WS-Policy [82]. WS-Policy is a metalanguage for specifying service characteristics and supported features so that service providers and consumers can coordinate their preferences (e.g. required security tokens, supported encryption algorithms, privacy rules).

WS-Trust [89] – defines a framework of trust models for establishing and managing direct or brokered trust relationships. Also allows for security tokens to be exchanged across different trust domains.

WS-Privacy [63] – defines a model for specifying the privacy preferences of service providers and consumers. At the time of writing, this specification has not been released.

The next level up is the Web services federation layer. This layer consists of the following specifications:

WS-SecureConversation [87] – describes how message exchanges are authenticated and managed. It covers security context exchange and establishing and deriving session keys.

WS-Federation [81] – defines a federation model for the federation of identity, authentication and authorisation information across security domains.

WS-Authorization [63] – for managing authorisation data and authorisation policies. At the time of writing, this specification has not been released.

Development on Web services security is currently still in progress. The completed components to date tend to be mainly directed at authentication and secure messaging. Despite this, the framework can be augmented with custom authorisation models to provide access control.

2.3 Access Control for Inter-organisational Workflows

Applying access control to inter-organisational workflows requires the design of an extended model. Inter-organisational workflows span across multiple organisations, presenting several challenges for access control enforcement. Recognising the challenges, Kang *et al.* [38] define the following access control requirements for inter-organisational workflows:

Separation of Application-level and Organisation-level Security – Changes to workflow participants (e.g. where a new organisation replaces an existing organisation in the workflow) should not require other organisations involved to restructure their security infrastructures in order to uphold the inter-organisational workflow. Hence inter-organisational workflows (i.e. the application-level security infrastructure) need to be insulated from organisation level changes to ensure that workflows can continuously operate without changing the workflow specifications.

Fine-grained Access Control – Inter-organisational workflows can contain many workflow tasks, with some tasks existing as threads within a process. Workflow tasks provide a user’s working context. Access control as far as the process level is not sufficient. A finer grained access control on the task level is needed to apply access control on the user’s working context.

Support for Dynamic Constraints – Dynamic constraints may be based on the users of a specific task (e.g. separation of duty constraints). Since inter-organisational workflows may consist of several autonomous workflows, a framework for workflow history sharing between participating workflows is required.

Warner and Atluri [77] focus on this third requirement by considering separation of duty constraints that span multiple instances of a workflow. Their work provides a formalism that includes predicates over temporal and workflow parameters so that “inter-instance” constraints can be specified as conditions on time and workflow history.

In a separate but related work, Warner *et al.* [78] investigate access control for inter-organisational resource sharing and propose an approach that exploits the “semantics associated with the user and object attributes associated with a specific role and its permissions”. An external user is granted access

to an object if he/she possesses at least the required set of attributes derived through the proposed process. This approach targets *dynamic* environments, characterised by entities joining or leaving the collaboration in an ad-hoc manner.

Kraft [41] lists several design goals for a distributed access control processor applicable to inter-organisational workflows. Such goals include extensibility, ease of integration, usability, performance and scalability.

Many security architecture proposals for inter-organisational workflows base their designs on web services [33, 40, 41]. This highlights the significance of web services as an enabling technology for such workflows. Although many web services technologies are open standards, such tightly coupled designs are too limiting. A broader model is needed that serves the access control requirements of inter-organisational workflows implemented using alternative enabling technologies.

2.4 Cryptographic Access Control

The concept of using cryptography for access control has been explored in several different contexts [1, 13, 17, 34, 46]. This access control paradigm attempts to abolish the notion of a reference monitor and relies exclusively on encryption to ensure the security of protected resources. It is well suited for providing distributed access control in untrusted (open) environments where there is a lack of global control. We look at a selection of the existing work in this field and briefly discuss their contributions and limitations.

Traditional Client/Server Architecture Example Harrington and Jensen [34] propose a cryptographic access control model to target large-scale open systems where no central authority exists. Data is stored encrypted on a server and retrieved by users on demand.

The model supports only read access and write access. Each protected object has a symmetric key (for encrypting the object) and a public/private keypair (for the object's digital signature). Users with read access require the symmetric key and public key, whereas users with write access require the symmetric key and the private key. The model ensures data integrity and availability by using a log-structured file system (modifications are written to a log instead of overwriting disk blocks). The server possesses each object's public key for verifying log entry signatures.

This proposal does not specify key distribution. Client machines are assumed to be the only trusted computing base in this model (the server and communication channels are open). Hence a malicious user could obtain the keys from a compromised client machine. Another limitation is that the model protects objects as a whole. Portions of an object cannot be separately protected.

Data Publishing Architecture Example Miklau and Suciu [46] propose a cryptographic access control framework for published data. It assumes that data (within the object) is organised into some hierarchical structure and it applies protections to individual portions of this structure. The data owner defines the access control policies over the object (policies are specified as queries). The queries are then evaluated to produce a logical data model, which is subsequently translated into a partially encrypted XML document. This document is then released for users to access.

The logical data model consists of a tree structure, where access to individual nodes is guarded by keys (or sets of keys). Keys are either exchanged via secure channel, stored in the XML document itself or exist as some data value (possibly stored in the XML document).

Permission types are not defined in the framework. User access is not subject to verification. Hence the proposal only considers confidentiality and not integrity. Key distribution is another issue. As keys are distributed by the data owner, this would not scale well for large-scale data dissemination. Furthermore, storing keys within the object risks overly inflating the object size (especially where many nodes exist) which would consequently negatively affect data dissemination.

Data Outsourcing Architecture Example Access control enforcement for data stored on external servers is explored by De Capitani di Vimercati *et al.* [17]. This work slightly extends on [46] by proposing an access control architecture that uses cryptography for protecting data in cases where data dissemination is outsourced to external servers.

The architecture defines two distinct repositories. One repository for storing (encrypted) resources and another for storing and managing access control policies. The architecture design allows different users to see different portions of the data resource (hence the resource, as a whole, cannot be encrypted with a single key). Each user is assigned (by the data owner) a secret key and a set of tokens (known as the *token catalog*). The token catalog represents the access control policy that the owner wants to enforce. Access to the data resource is protected using *resource keys* that are derived from the secret

key and token(s) issued by the data owner.

This architecture introduces an improved key solution to [46] but key distribution (and additionally, token distribution) is still the responsibility of the data owner. Another drawback affecting scalability is that each user requires a separate secret key from each data owner. The proposal targets *read-mostly* resources so no distinction is made between read access and write access.

2.5 Summary

Current research into access control for inter-organisational workflows and cryptographic access control have encompassed many forms, a few of which have been presented in this literature review. However, these works have predominately focused on the theoretical side and less on the practical aspects. Hence on a practical level, combining inter-organisational workflow, security policies and access control is not well understood.

This thesis aims to use cryptographic access control in the context of inter-organisational collaborative environments and focus on how this form of access control can be applied to satisfy the security requirements of users in such environments. Previous studies into access control for inter-organisational environments have identified several needs to be addressed. The following chapter explains the architecture model proposal of this thesis, detailing how we apply cryptographic access control to inter-organisational workflows.

Chapter 3

Architecture Model

This chapter presents the proposed architecture model, including its features and message exchanges, to facilitate access control for inter-organisational information flow. This model is intended to be generic by design so that it can be readily applied to various applications. As such, it is not targeted towards any specific industry or application.

3.1 Overview

The architecture model serves as the platform for supporting workflows in inter-organisational collaborative environments. We assume that each workflow contains a single *workflow object* that is circulated amongst numerous *workflow participants* in some well-defined sequence. The architecture model can be defined as follows:

“A generic platform that provides decentralised access control and verification for inter-organisational information flow.”

The model categorises workflow participants into two types: *ordinary workflow participants* and *verifying workflow participants*. An ordinary workflow participant performs actions that are subject to verification. In other words, ordinary workflow participants are not fully trusted to perform only legitimate actions on the workflow object. A verifying workflow participant is responsible for checking the workflow object once it has passed through an ordinary workflow participant. For example, it may

check data values within the workflow object assigned by the previous ordinary workflow participant. The verifying workflow participant is assumed to be trusted not to perform illegitimate actions.

Both types of workflow participants are considered as *recipients* of the workflow object. The terms workflow participants and recipients are used interchangeably. Once a workflow object has been instantiated, it can only be circulated between recipients. An *issuing authority* instantiates a new workflow object and releases it to the first recipient in the workflow.

The architecture model only supports pre-defined workflows. Each workflow object has a *workflow description* that defines the valid flow paths that the workflow object may take as it is circulated amongst recipients. Flow paths that are not specified in the workflow description cannot be legitimately followed during the lifetime of the workflow object. Essentially, the workflow description describes the flow control rules that the workflow object must adhere to. Control over the design and structure of the workflow object (including the workflow description) rests with the issuing authority.

The issuing authority uses the workflow object's schema to instantiate a new instance of the workflow object (which is subsequently released to the first recipient of the workflow). Although the issuing authority defines the flow control rules for the workflow object, it does not necessarily define all of the access control rules for it. Access control is enforced via encryption techniques as the architecture model does not depend on a central authority for providing access control enforcement. Because of this, encryption keys are needed to both secure and to provide access to protected parts of the workflow object. The release of encryption keys in the architecture model is controlled by an *identity provider*. The identity provider knows the users operating within the same administrative domain, as well as their public keys.

A simplified view of how the above mentioned entities interact in the architecture model is given in Figure 3.1. A new workflow instance is started with the issuing authority:

1. instantiating a new workflow object
2. creating a shared secret (between itself and the identity provider)
3. initialising the new workflow object
4. releasing the new workflow object to the first recipient specified in the workflow description

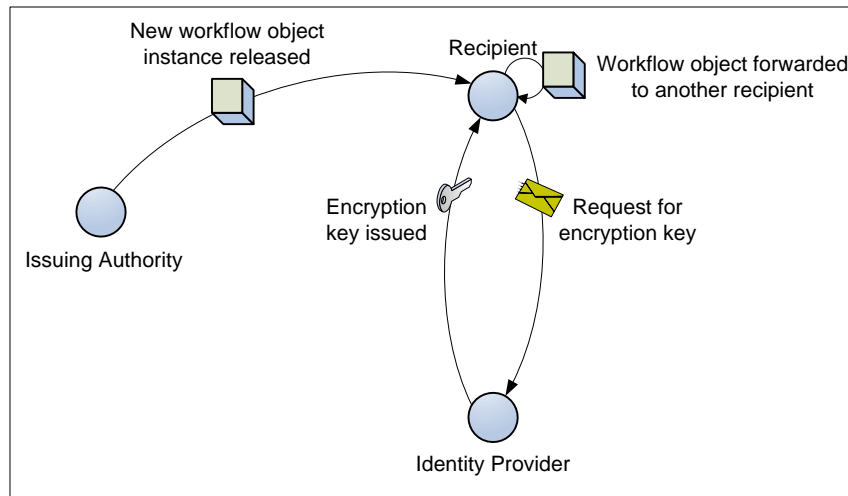


Figure 3.1: Simplified view of the architecture model.

If the recipient needs a key to access a protected component of the workflow object, it must send a request to the identity provider. Provided that the recipient is permitted to receive the key, the identity provider proceeds with issuing the required key to the recipient.

Once the recipient has finished processing the workflow object, the workflow is continued with the recipient forwarding the object to the next recipient in the workflow description. Depending on the workflow description, the next recipient may be a verifying workflow participant (assigned to check the changes made to the object by the first recipient) or another ordinary workflow participant. As before, the new recipient requests any required keys from the identity provider and forwards the object to the next workflow participant after processing. This procedure continues until the workflow ends.

In summary, the main steps of the architecture model are:

1. The recipient receives the workflow object.
2. If the user is authorised to access protected components of the workflow object, the identity provider releases the appropriate keys upon request.
3. After the recipient has processed the workflow object, it may be necessary for the workflow object to be verified by a verifying workflow participant. This is specified in the workflow description, as well as any recovery steps for when the workflow object does not pass verification.

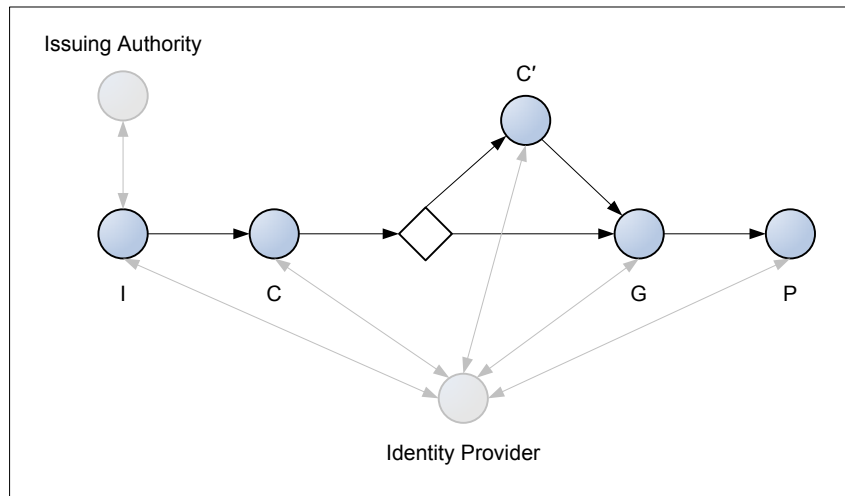


Figure 3.2: Example workflow used for subsequent explanations of the architecture model.

4. The workflow object history is updated and the workflow object is transferred to the next recipient in the workflow.

3.2 Example Workflow

In the remaining sections of this chapter, we use the following example to explain the different concepts of the architecture model. This example workflow is based on a purchase ordering scenario.

The workflow starts with an order being made for the purchase of some goods (or services). This activity is performed by the workflow participant *I*. Next, the order is checked and approved by a separate workflow participant *C*. At this point, the order may require additional checking or approval (e.g. from a higher authority). Let us denote this second checker as *C'*. Once the order has been approved, it is fulfilled by *G*. Finally, payment for the order is handled by *P*. Figure 3.2 illustrates this workflow. The diamond in this figure represents a decision point (in this case, the flow takes one of two possible paths).

For this workflow, there is a workflow object (representing the purchase order) that is forwarded and processed in turn by each workflow participant. Using this example, we demonstrate how the architecture model can be applied to secure the data within this object throughout the workflow.

The dimmed parts in Figure 3.2 show those components specific to the architecture model. These components are the issuing authority and the identity provider. The issuing authority releases a new workflow object to I . At this point in the workflow, the workflow object can be imagined as a blank purchase order form (that is subsequently filled out by I). Both the issuing authority and the identity provider are known to all workflow participants in advance. The identity provider also knows the identity of all workflow participants (as well as their public key). In this workflow, I , G and P are ordinary workflow participants, whereas C and C' are verifying workflow participants. For brevity, verifying workflow participants are not provided for G and P .

3.3 User-based Model

One approach is to realise all workflow participants as individual users. Using the example workflow, this would mean that I , C , C' , G and P are all users and their identities are known beforehand (i.e. they are explicitly defined in the workflow description). As encryption plays a pivotal part in this architecture model, an important aspect to consider is key exchange. Encryption keys need to be securely exchanged so that authorised users can decrypt (or verify) protected items of the workflow object. A user-based model helps to simplify this task as all users already have a public/private keypair. As such, encryption keys need not be exchanged as user keypairs can be used instead.

For example, suppose I chooses to protect an item in the workflow object for C . This can be achieved as follows:

$$\{\{data\ item, id_{WF}\}_{signed\ by\ I}\}_{encrypted\ for\ C}$$

Should C need to reveal this protected item to C' , the item can be re-encrypted as follows:

$$\{\{\{data\ item, id_{WF}\}_{signed\ by\ I}\}_{encrypted\ for\ C'}\}$$

In the above examples, the signatures are created with the private key of I and the encryption is performed with the public keys of C and C' respectively. Notice also that the protected item includes a workflow instance identifier (id_{WF}) to avoid replay attacks (i.e. substituting the protected item with a past protected item also signed by I). In principle, more complex schemes can be applied to protect against other types of attacks.

This approach may seem simple enough for the purposes of secure information flow, but problems arise if one of the users change. If we could assume that, for every workflow instance, the same users would always participate in the workflow, the user-based approach would be fine. But in reality, user change will inevitably occur. Because of this, whenever a user changes, the workflow description must be changed. This is necessary even if the user change is temporary. As a result, this makes user administration difficult. The workflow description should rarely be modified, but with a user-based model, frequent modifications to the workflow description would be required.

3.4 Role-based Model

An alternative approach is to view workflow participants as roles. A role describes a job function (as opposed to the identity of the person performing the task). For service-oriented environments, if we imagine each task in the workflow being implemented by a service, then roles are the equivalent to service descriptions. That is, a service description describes the type of service required to fulfil the task, and a service satisfying this service description is selected to perform the task.

Let us now consider the workflow participants in the example workflow in Section 3.2 as roles rather than users. A role can be played by one or more users. Also, an individual user can be assigned to one or more roles. Now that users are removed from the workflow description, this resolves the problem highlighted in the user-based approach (Section 3.3). Roles are far less likely to change in comparison to the users participating in the workflow. Although role-related modifications to the workflow description may be relatively rare, user-role assignment changes are more frequent (i.e. specifying which users are authorised to perform which roles). However, user-role assignment is not specified in the workflow description as user administration is the responsibility of the identity provider. Hence the identity provider must maintain a table specifying which users are assigned to which roles.

From the example workflow, although the roles I , C , C' , G and P are all listed in the workflow description (and therefore known beforehand), the actual users participating in the workflow are progressively decided upon during runtime. But as users can be assigned to multiple roles, *separation of duty* becomes an important issue. An example separation of duty rule could state that a user cannot

play role C' after previously playing C in the same workflow execution. Such rules are specified in the schema and are enforced by the identity provider. In principle, additional issues, other than separation of duty, can also be addressed.

3.5 Workflow Instantiation

The workflow instantiation procedure was briefly introduced in Section 3.1. As mentioned, this procedure mainly involves the issuing authority. Workflow instantiation is triggered when the issuing authority receives a request for a new workflow object. This request may be sent by the first recipient in the workflow description or by some other authorised entity. After receiving this request, the issuing authority proceeds with instantiating a new workflow object. Each new workflow object is identified by a unique ID (id_{WF}) (previously introduced as the workflow instance identifier in Section 3.3). This value is created by the issuing authority and is formed as follows:

$$id_{WF} = Enc_{pubKey_{idP}}\{timestamp, secret\}$$

where $secret = rand()$

The id_{WF} needs to be sufficiently large (e.g. > 100 bits) to maximise the probability that uniqueness is maintained over the life of the system. A timestamp (indicating the creation time) and a secret (representing some random value) are encrypted using the identity provider's public key to produce the workflow ID. The secret becomes a shared secret between the issuing authority and the identity provider. The timestamp is included in case the same secret is ever used on two or more occasions. This way, one cannot deduce that a secret has been used more than once by simply observing the workflow ID. This is important as the timestamp and secret are later used in the creation of symmetric keys.

In the next step of the workflow instantiation procedure, the issuing authority initialises the new workflow object with any required values. For example, the newly calculated workflow ID is assigned to the object. Depending on the type of object, there may be other values that must be set during initialisation (e.g. for a purchase order, a purchase order number must be set). The schema defines those values that the issuing authority must initialise.

Once initialisation is complete, the new workflow object is released to the first recipient specified in the workflow description. But the workflow description only lists the role of the first recipient and not an actual user. In the case where the workflow instantiation was triggered by the first recipient, the workflow object can be directly released to this user. However, if the issuing authority does not know a specific user to release the object to, it must send a request to the identity provider to fetch a user matching the role of the first recipient. The identity provider replies with the identity of an authorised user. Alternatively, the identity provider may provide a list of authorised users for the issuing authority to choose from.

3.6 Key Exchange

In Section 3.3, key exchange is mentioned as an important aspect of the architecture model. An advantage of the user-based model approach is that each user's public/private keypair can be used for encrypting protected items in the workflow object, hence eliminating the need for key exchange. But the same cannot be done for the role-based model as public/private keypairs are assigned to users, not roles. To overcome this limitation, shared keys are exchanged.

Shared keys in the architecture model are assigned to roles. They can be assigned in the following ways:

1. for a single role - e.g. role G
2. for a set of roles - e.g. roles $\{C, G, P\}$
3. for a role instance - e.g. role instance G_2

A key assigned for a single role can only be issued to users belonging to that role. A key assigned for a set of roles can only be issued to users belonging to at least one of the roles defined in the set. A key assigned for a role instance can only be issued to the user enacting the role instance. In other words, suppose that the workflow description includes a loop and a task within that loop is assigned to role G . The role instance G_2 indicates the second occurrence of role G in the workflow execution. Similarly, in a sequential flowpath where there are two different tasks and each task is assigned to role

G , then G_2 indicates the role in the latter task as this would be the second occurrence of role G in the workflow execution.

Shared keys are created by the identity provider. For example, a shared key for roles $\{I, G\}$ is formed as follows:

$$k_{\{I,G\}} = f(\text{subject}, \text{timestamp}, \text{secret})$$

$$\text{where } \text{subject} = \{I, G\}$$

As shown, the shared key is the result of a function that accepts three values: *subject*, *timestamp*, *secret*. It must not be possible to reverse this function (i.e. to extract these three values from the shared key). For example, this could be a hash function such as MD5 [55]. The subject indicates to whom the key is assigned to. The timestamp and the secret are extracted from the workflow ID (id_{WF}). This is achieved as follows:

$$\{\text{timestamp}, \text{secret}\} = Dec_{privKey_{idP}}(id_{WF})$$

Apart from being linked to roles, shared keys are also linked to workflow instances (as indicated by the use of the values concealed in the workflow ID).

The identity provider distributes shared keys to authorised users. When a user requires a shared key, it sends a request to the identity provider. Using the example workflow from Section 3.2, consider the shared key $k_{\{I,G\}}$ that is shared between roles $\{I, G\}$. User g (who is a user of role G) is currently executing the workflow task assigned to role G . For g to obtain $k_{\{I,G\}}$, it must send a request to the identity provider. This request is formed as follows:

$$\text{request} = \{\text{subject}, id_{WF}\}_{Sig_g}$$

$$\text{where } \text{subject} = \text{the subject of the desired shared key - i.e. } \{I, G\}$$

The request is signed by g (using its private key) so that the identity provider can verify from whom the request is from. The identity provider must also verify whether g is authorised to receive the desired shared key. In the case where the subject is a role instance, the workflow object's history (discussed later in this chapter) must also be sent with the request. If the user is a legitimate subject of the key, the identity provider calculates the shared key (from the data contained in the request) and issues it to the user. In this case, the key is issued to user g as follows:

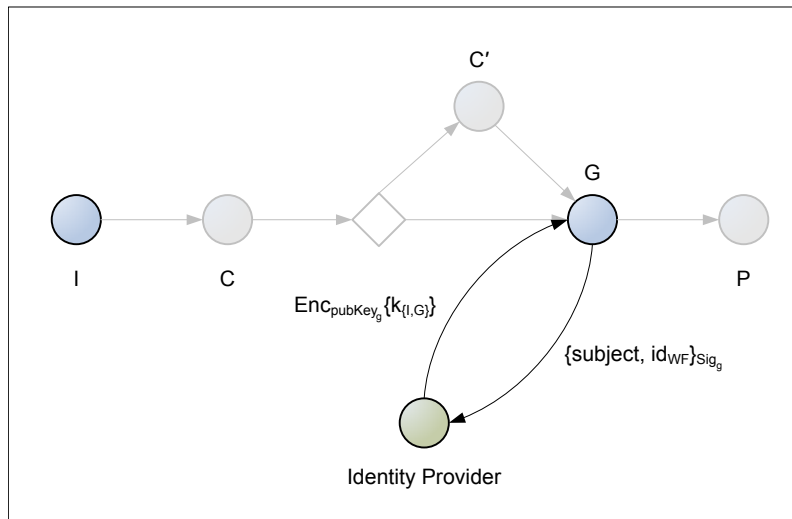


Figure 3.3: Request/response procedure for obtaining shared key.

$$Enc_{pubKey_g} \{k_{\{I,G\}}\}$$

By encrypting the shared key with the public key of g , the shared key can be securely issued. User g simply decrypts the response from the identity provider (using its private key) to reveal the shared key. The shared key can now be used by g to decrypt any protected items in the workflow object that have been secured with this key. This example is illustrated in Figure 3.3.

3.6.1 User-defined Shared Keys

An alternative to the identity provider creating keys is for the users themselves to create shared keys. In this situation, a shared key is assigned to one or more protected data items in the workflow object. The user who first protects the data item(s) is responsible for creating the shared key. Although the key is created by a user, it is issued to other authorised users by the identity provider (via a similar request/response procedure as discussed in Section 3.6). This is achieved using key indexes.

To explain, consider the scenario discussed in Section 3.6 where there exists a shared key for roles $\{I, G\}$. This key is for protecting a data item that can only be disclosed for users of roles $\{I, G\}$. The user playing role I (e.g. user i) creates the shared key since this user processes the workflow object before G . This key is formed as below:

$$k_{\{I,G\}} = f(\text{subject}, \text{secret}_i, \text{id}_{WF})$$

where $subject = \{I, G\}$

$secret_i = rand()$

To create this key, the user calculates a random value ($secret_i$) that should only become known to the identity provider. Inclusion of the workflow ID is to ensure confidentiality in case the same random value is used for a shared key in a different workflow object. The identity provider learns of the shared key through the key index for that key. A key index is calculated as below:

$$index_{k_{\{I,G\}}} = Enc_{pubKey_{idp}}\{secret_i, id_{WF}\}$$

This index is stored in the workflow object as metadata for the data item protected with $k_{\{I,G\}}$. When control of the workflow object eventually passes to a user of role G (e.g. user g), this user requests the shared key from the identity provider. The request is formed as below:

$$request = \{index_{k_{\{I,G\}}}, subject\}_{Sig_g}$$

where $subject = \{I, G\}$

The identity provider decrypts the key index to obtain $secret_i$ and id_{WF} . Additionally, it verifies that the signer is an authorised subject. If the user is deemed authorised to receive the key, the identity provider forms the shared key using the data supplied by the requester. The key is securely issued to user g in the same fashion as described in Section 3.6:

$$Enc_{pubKey_g}\{k_{\{I,G\}}\}$$

Regardless of whether the identity provider creates shared keys or the user, authorised users can replace (overwrite) an existing shared key with a new key. Depending on the circumstances of the protected item, this may be permissible. However, in cases where key replacement is prohibited, it is the responsibility of a verifying workflow participant to check that the correct keys are used. In other words, that the key subject is valid.

3.7 Access Types

The architecture model supports the following types of access to items protected in the workflow object:

1. Read only access
2. Read write access
3. Append access

Each access type is enforced via encryption. The following three sections discuss each access type in detail. For the descriptions in each section, assume that the data item under protection is protected by i and access is granted only for roles $\{I, G\}$.

3.7.1 Read Only Access

This access type allows authorised users to read protected data but not edit this data. Let $data$ represent a data item with read only access. This data item is protected as follows:

$$\{Enc_{k_{\{I,G\}}} \{\{data\}_{sig_i}\}\}_{sig_{k_{\{I,G\}}}}$$

The data item is first signed by the protector (i.e. user i) using their private key. Any modifications made to the data item by authorised users can be detected from the broken signature. The signed data item is then encrypted using the shared key issued to authorised users (i.e. $k_{\{I,G\}}$). This prevents unauthorised users from reading the data item. Finally, the encrypted data is signed using the shared key. Any modifications made by unauthorised users can be detected from the broken outer signature.

3.7.2 Read Write Access

This access type allows authorised users to read and edit protected data. Let $data$ represent a data item with read write access. This data item is protected as follows:

$$\{Enc_{k_{\{I,G\}}} \{data\}\}_{sig_{k_{\{I,G\}}}}$$

Read write access protection is the same as for read only access protection except that it does not contain an inner signature. Encrypting the data item with the shared key ensures that only authorised users are granted access. As for read only access, the signature formed with the shared key is for detecting any modifications made by unauthorised users. To verify modifications made to the data item by authorised users, policies are required. Such policies express the data validation rules that

apply to the data item. With all access types, verification is the responsibility of verifying workflow participants.

3.7.3 Append Access

The architecture model recognises two types of append access. These types are:

1. Can read existing value; append new value
2. Cannot read existing value; append new value

Let us first consider the case where a user can read the existing value when appending new data to the data item. The data item is initially protected as:

$$\{Enc_{k_{\{I,G\}}} \{\{data\}_{Sig_i}\}\}_{Sig_{k_{\{I,G\}}}}$$

The existing value is signed by user i (the initial protector) to make it write protected to other users. This signed value is then encrypted with the shared key $k_{\{I,G\}}$ to ensure that only users of role I or G are granted access to the data item. Finally, the encrypted item is signed with the same shared key to detect any modifications performed by unauthorised users.

Eventually, possession of the workflow object passes to user g . Suppose that this user decides to append the value x to the protected data item. When this action is performed, the data item is secured as follows:

$$\{Enc_{k_{\{I,G\}}} \{\{\{data\}_{Sig_i}, x\}_{Sig_g}\}\}_{Sig_{k_{\{I,G\}}}}$$

The user appends the value x to the existing signed value, and then signs this combined value with its own private key. To explain more clearly, let $\{\{data\}_{Sig_i}, x\} = data'$. The above expression can now be simplified to:

$$\{Enc_{k_{\{I,G\}}} \{\{data'\}_{Sig_g}\}\}_{Sig_{k_{\{I,G\}}}}$$

Note the similarity between this expression and the first one from user i . Imagine if another user of role G (say user \tilde{g}) was to append to this data item now (although such an action violates the workflow description described in Section 3.2). For example, suppose \tilde{g} appends the value x' . The result of this action is:

$$\{Enc_{k_{\{I,G\}}} \{ \{ \{ data' \}_{Sig_g, x'} \}_{Sig_{\tilde{g}}} \} \}_{Sig_{k_{\{I,G\}}}$$

Again, we can apply the same simplification as previously (i.e. let $\{ \{ data' \}_{Sig_g, x'} \} = data''$), and hence we continue to see the same pattern emerging.

Let us now consider the case where a user cannot read the existing value when appending new data to the data item. In this case, the initial protector of the data item must be a trusted user. This is because the initial user can read and edit the existing value without detection. This privilege does not extend to any other users. The data item is initially protected as:

$$\{Enc_{pubKey_i} \{data\}\}_{Sig_{k_{\{I,G\}}}$$

The data item is first encrypted with the public key of the trusted user (in this example, user i). This prevents the existing value from being read by other users. The encrypted value is then signed with the shared key $k_{\{I,G\}}$ to ensure that only users of role I or G (i.e. authorised users) can append to the data item.

As before, when possession of the workflow object passes to user g , this user decides to append the value x to the protected data item. The data item is secured as follows:

$$\{Enc_{pubKey_i} \{Enc_{pubKey_i} \{data\}, x\}\}_{Sig_{k_{\{I,G\}}}$$

User g cannot read the existing value (i.e. $data$) as this is secured with i 's public key. The value x is simply appended to the existing value and then g encrypts this combined value using i 's public key. Similarly to the first append type, let $\{Enc_{pubKey_i} \{data\}, x\} = data'$. The above expression can now be simplified to:

$$\{Enc_{pubKey_i} \{data'\}\}_{Sig_{k_{\{I,G\}}}$$

Again, there is a similarity between this expression and the initial one. Continuing from the example with the first append type, suppose that now user \tilde{g} appends the value x' to the data item. The data item now appears as:

$$\{Enc_{pubKey_i} \{Enc_{pubKey_i} \{data'\}, x'\}\}_{Sig_{k_{\{I,G\}}}$$

Consequently, there is also a pattern emerging with this append type. We can simplify this latest append action to:

$$\{Enc_{pubKey_i}\{data''\}\}_{Sig_{k_{\{I,G\}}}}$$

where $data'' = \{Enc_{pubKey_i}\{data'\}, x'\}$

3.8 Workflow Object History

The workflow object history lists, in order of receipt, the users who have previously possessed the workflow object. It does not list specific modifications that users have made to the object. Instead, it indicates the object's location in the workflow and the flow path taken by the object to reach its current point.

In the workflow description, workflow participants are identified by roles. Hence, it is unknown which user will enact the role until the previous recipient is ready to release the workflow object. Effectively, the workflow description describes the future for the workflow object. In contrast, the workflow object history describes the past. It allows us to check which users enacted which roles up to the current point in the workflow. This helps ensure that the workflow description has been correctly followed. It also forms an audit trail of recipients. As such, the workflow object history serves to make users more accountable for their actions on the object (and hence avoid authorised users from abusing or misusing their privileges). Although actions are not recorded, verifying workflow participants can check the history to determine the user(s) responsible for invalid or incorrect data in the workflow object.

Recording the workflow object history starts with the issuing authority. As part of the workflow instantiation procedure, the issuing authority signs the *constraints* bound to the workflow object. Constraints include the workflow description and the workflow ID. The object itself is also signed if the issuing authority adds its own values to the object. The initial history is represented as:

$$\{constraints, [object]\}_{Sig_{IAuth}}$$

Assuming user i is the first recipient, this user verifies the signature before processing the workflow object. Once i has finished, it signs the object together with the previous signature (in this case, the

issuing authority's signature) before forwarding the object to the next recipient. At this point, the workflow object history appears as:

$$\{\{constraints, [object]\}_{Sig_{IAuth}}, object'\}_{Sig_i}$$

where $object'$ = the object after being processed by user i

The next recipient is user c . This user checks i 's signature to ensure that the object is not corrupted. As before, once c has finished, it signs the object together with the previous signature (this time, i 's signature) before forwarding the object to the next recipient. The workflow object history now appears as:

$$\{\{\{constraints, [object]\}_{Sig_{IAuth}}, object'\}_{Sig_i}, object''\}_{Sig_c}$$

where $object''$ = the object after being processed by user c

This cycle continues for the remainder of the workflow.

An alternative approach for recording the workflow object history relies on identity providers to produce the signatures. Identity providers are used as they are *trusted entities* in the architecture model. This alternative approach works as follows:

$$h \in WorkflowObjectHistory$$

$$idp_1 \in IdentityProvider$$

$$r_1 \in Role$$

$$u_1 \in User$$

$$\text{let } h = \{v\}_{sig_{idp_j}}$$

where idp_j = the identity provider who last signed the workflow object history

v = the workflow object history value

$$u_1 \rightarrow idp_1 : h, ACQ$$

$$idp_1 \rightarrow u_1 : \{h, (u_1, r_1, ACQ)\}_{Sig_{idp_1}}$$

pre: h is valid \wedge last record in h is not ACQ

post: $\{h, (u_1, r_1, ACQ)\}_{Sig_{idp_1}} = h'$

$$u_1 \rightarrow idp_1 : h', REL$$

$$idp_1 \rightarrow u_1 : \{h', (u_1, r_1, REL)\}_{Sig_{idp_1}}$$

pre: h' is valid \wedge last record in h' is not REL

The user u_1 , enacting role r_1 , belongs to the domain of idp_1 . Before processing the workflow object, u_1 informs its local identity provider (i.e. idp_1) that it has acquired the workflow object.

The user sends the existing history h to the identity provider, together with an *acquire* flag (ACQ). The identity provider creates a new history record – $\{h, (u_1, r_1, ACQ)\}$ – and signs this record. The signed history record is sent to the user to be appended to the object metadata.

After processing the workflow object, u_1 informs idp_1 that the workflow object is ready to be released (i.e. passed on to the next user in the workflow). Similarly to before, u_1 sends the workflow object history and a *release* flag (REL) to idp_1 . The identity provider creates a new history record – $\{h', (u_1, r_1, REL)\}$ – and signs this record. The signed history record is sent to the user to be appended to the object metadata.

3.9 Multiple Domains

Until now, the examples in this chapter have focused on a single domain. In other words, we have assumed that all workflow participants operate within the same security realm with a commonly recognised identity provider. But an important aspect of this research is its support for *inter-organisational workflows*. Hence, we must also consider workflows that span across several domains - each with their own identity provider.

Introducing multiple identity providers to the workflow requires changes to some of the previously discussed encryption expressions that assume a single identity provider (e.g. the workflow ID). To

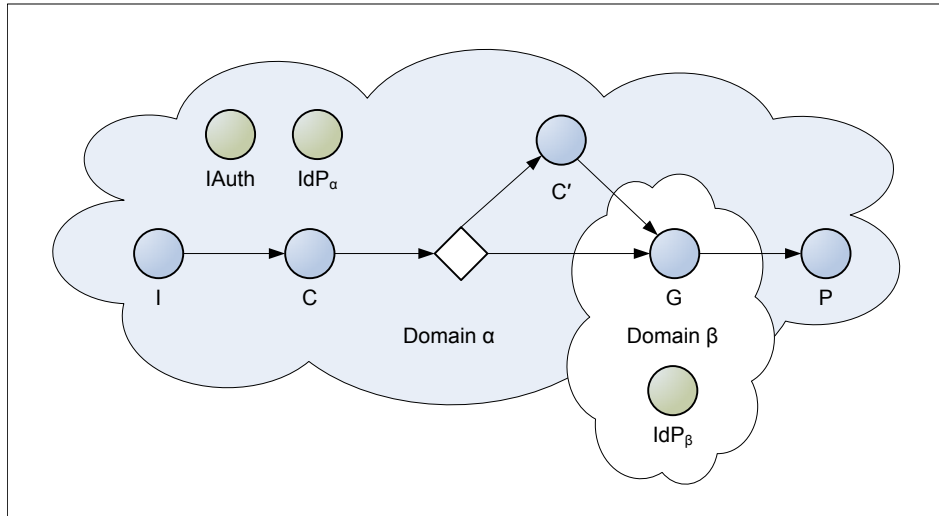


Figure 3.4: Example workflow modified to include multiple domains.

explain, let us first modify the workflow example in Section 3.2 to contain multiple domains with separate identity providers for each domain. This modification is illustrated in Figure 3.4. In this diagram, G is in a separate domain from all other workflow participants. All workflow participants except G use IdP_α as their identity provider, whereas G uses the identity provider IdP_β . Now that G is in a separate domain, this role is solely administered by IdP_β . This means that a role G administered by IdP_α would be a different role altogether. In other words, roles are relative to identity providers.

The two procedures affected by the introduction of multiple identity providers are *workflow instantiation* and *key exchange*. For workflow instantiation, creating a new workflow ID is affected. Recall from Section 3.5 that the workflow ID is formed by encrypting the timestamp and secret with the identity provider's public key. This formula cannot be used for inter-organisational workflows as the workflow ID can only be decrypted by a single identity provider. Hence, a common key is required between all identity providers so that each can decrypt the workflow ID. Although we assume that each identity provider has its own public/private keypair, we cannot assume that a shared key already exists between all the identity providers involved in the workflow.

We solve this problem by using *tokens* allocated to each identity provider. Firstly, the workflow ID is created by encrypting the timestamp and secret with a shared key k . This is expressed as:

$$id_{WF} = Enc_k\{timestamp, secret\}$$

where $k = rand()$

The timestamp and secret are created the same as before. Similarly to the secret, the shared key k is also a random value created by the issuing authority. The issuing authority also creates a token for each identity provider involved in the workflow. In the example workflow, two tokens are created (i.e. $token_{IdP_\alpha}$ and $token_{IdP_\beta}$). These are formed as follows:

$$\begin{aligned} token_{IdP_\alpha} &= Enc_{pubKey_{IdP_\alpha}}\{k, id_{WF}\} \\ token_{IdP_\beta} &= Enc_{pubKey_{IdP_\beta}}\{k, id_{WF}\} \end{aligned}$$

Tokens are unique for each workflow instance. In case identical values for k are used in two or more workflow instances, we include the workflow ID inside the token to ensure uniqueness. By decrypting their respective token, an identity provider learns of the shared key k so that they can subsequently decrypt the workflow ID to extract the timestamp and secret.

For key exchange, the request/response procedure discussed in Section 3.6 changes slightly. In the request message, the workflow ID is replaced with the token of the identity provider to whom the request is being sent. For example, user g (of role G) sends the following request message to its identity provider:

$$request = \{subject, token_{IdP_\beta}\}_{Sig_g}$$

As the token includes the workflow ID, the identity provider can easily extract this information and proceed as before with issuing the key to the user. The response message from the identity provider remains unchanged.

For user-defined shared keys, multiple key indexes may be required. If the key is to be shared between roles of multiple identity providers, an index for each identity provider is needed. Using the same example from Section 3.6.1, the shared key $k_{\{I,G\}}$ has the following indexes:

$$\begin{aligned} index_{\alpha_{k_{\{I,G\}}}} &= Enc_{pubKey_{IdP_\alpha}}\{secret_i, id_{WF}\} \\ index_{\beta_{k_{\{I,G\}}}} &= Enc_{pubKey_{IdP_\beta}}\{secret_i, id_{WF}\} \end{aligned}$$

The request/response procedure for user-defined shared keys remains largely unchanged. The only exception is that the key index in the request message is replaced with the specific key index for the

identity provider to whom the request is being sent. For example, user g uses the key index $index_{\beta_{k_{\{I,G\}}}}$ in its request message to IdP_{β} . Hence, the request message appears as follows:

$$request = \{index_{\beta_{k_{\{I,G\}}}}, subject\}_{sig_g}$$

3.10 Multiple Permissions

So far the architecture model describes how data items are protected with only a single access type. But what if we want to apply multiple access types to the one data item? For example, allow *read only* access to role G and *read write* access to role C .

One solution is to duplicate the data item and then protect each copy separately (i.e. one protected for role G and the other protected for role C). The problem with this is how to ensure that changes to the data item are consistently propagated to all copies? Duplicating data items is not an ideal solution as it introduces unnecessary complexity and does not scale well.

Another solution is to rely on trusted entities (e.g. identity providers) to guard protected data items and release them to authenticated users on demand. This complicates the architecture model by separating protected data items from the workflow object. Furthermore, this approach is a step towards a centralised architecture, hence conflicting with the objectives of this research.

A better solution (which is more aligned with the previously discussed features of the architecture model) is to protect the data item with multiple shared keys. This requires an encryption technique that accommodates each permission assigned to the data item.

Let us consider the following example where the data item is protected for both *read only* and *read write* access:

$$\{Enc_{k_1} \{ \{data\}_{sig_{k_2}} \} \}_{sig_{k_1}}$$

where $k_1, k_2 \in SharedKey$

Roles assigned *read write* permission require both keys (i.e. k_1 and k_2) to access the protected data item. On the other hand, roles assigned *read only* permission must only have k_1 . Hence k_1 enables the data item to be viewed and k_2 enables the data item to be modified.

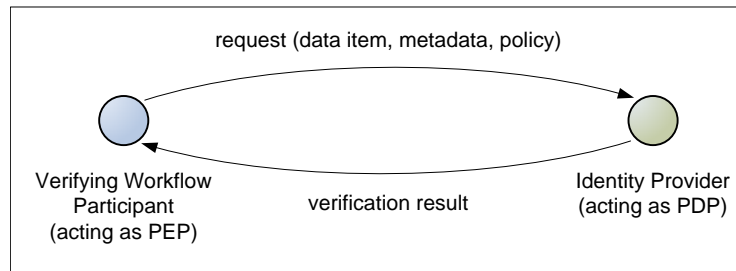


Figure 3.5: Architecture extension to support verification by identity providers.

The inner signature (created with k_2) exists to detect unauthorised modifications to the data item. The data item and the inner signature are encrypted with k_1 to conceal the data item from unauthorised roles. The resulting ciphertext is signed with k_1 to detect unauthorised modifications to the data item.

In this example, we have only considered data items protected for *read only* and *read write* access. Alternative permission type sets would require different encryption techniques to be proposed.

3.11 Identity Provider as Verifier

In Section 3.1, the distinction between ordinary workflow participants and verifying workflow participants is made. The model cannot prevent ordinary workflow participants from executing prohibited actions. Hence, a verifying workflow participant must verify the workflow object after an ordinary workflow participant has processed it. This is to ensure that the workflow object has not become corrupted from illegitimate actions.

But what if a *protected* data item needs to be checked and the verifying workflow participant is not authorised to access this item? In this case, the verifying workflow participant cannot verify whether data item satisfies the policy rules.

For these situations, identity providers serve as verifiers. Identity providers are well suited since they are trusted entities in the architecture model and they can derive shared keys. Verification is regarded as an added function to the identity providers' main activities of providing identity management and key distribution.

To support verification by identity providers, the architecture model adopts an approach largely influenced by the XACML architecture model [20]. The design is illustrated in Figure 3.5. Using

XACML terminology, the verifying workflow participant represents the Policy Enforcement Point (PEP) and the identity provider represents the Policy Decision Point (PDP). The verifying workflow participant supplies the identity provider with the protected data item, the data item's metadata and the policy rule(s) to evaluate. As such, the verifying workflow participant can also be regarded as the Policy Access Point (PAP) and the Policy Information Point (PIP).

The identity provider derives the required shared key to unprotect the data item (the metadata must contain sufficient information to achieve this), checks the data item value against the policy rule(s) and returns the verification result to the verifying workflow participant. In case the identity provider requires additional data within the workflow object to evaluate the policy rule(s), the entire workflow object may be sent rather than just the protected data item. Similarly to XACML, the identity provider may respond with a verification result of *permit*, *deny*, *not applicable* or *indeterminate*. Once the verifying workflow participant receives the result, it then enforces this decision.

3.12 Summary

This chapter has defined a role-based, cryptographic approach for securing information flow in inter-organisational collaborative environments. The proposed architecture protects sensitive data contained in a workflow object (an example being a document with many contributors) that follows a pre-defined workflow. Access to protected data items in the workflow object depends on possession of shared encryption keys and role enactment.

The architecture model realises several trusted entities. One of which, identity providers, are responsible for authenticating users and for distributing shared keys to authorised users. Verifiers are also considered as trusted entities and are responsible for verifying the workflow against security policies at various stages throughout the workflow.

In the next chapter, we look at the workflow object in detail, focusing on the components forming this object. Although the cryptographic functions used in the architecture model (i.e. *Enc*, *Dec*, *Sig*) are standard, we apply them to data and metadata contained in the workflow object.

Chapter 4

Workflow Object Model

After specifying the architecture model in the previous chapter, we now turn our attention to the workflow object's design. This chapter describes the components forming the workflow object model and its relationship to the architecture model. Although the architecture model touches on some aspects of the workflow object, this chapter concentrates on the finer details.

4.1 Components

The workflow object model is comprised of the following five components: *object data*, *object metadata*, *schema*, *policies* and *template*.

Figure 4.1 shows how these components are connected. The object data and object metadata components form the core of the workflow object. Constraints applied to the workflow object are expressed in the schema and policies components. Using the template component, the *object view* is produced. This provides a user-friendly, human readable view of the workflow object.

The *object data* component contains a structured collection of data items. A data item can be likened to a data field with a data value assigned to it (in other words, a name-value pair). Alternatively, a data item may contain other data items. This allows a data hierarchy to be formed. This component can be conceptually considered as a graph, with the data items forming the nodes of the graph. The structural design of the object data component is inspired by the W3C's Document Object Model [18].

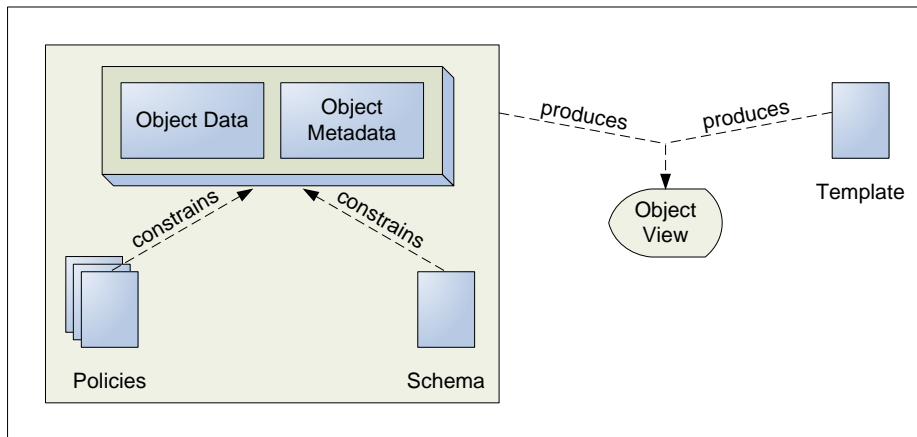


Figure 4.1: The workflow object model components.

The *object metadata* component contains metadata for the data items located in the object data component. A significant portion of this component is cryptographic-related information. For example, this may include encryption details (such as the encryption algorithm used), key retrieval details, digital signatures, etc. The object data component categorises its data items as either protected or unprotected. Unprotected data items are freely accessible to all recipients of the workflow object (and therefore require no data protection). On the other hand, a protected data item requires metadata so that authorised users can unprotect and access it.

The *schema* and *policies* components express constraints applied to the workflow object. Supported constraints can be broadly categorised into four different categories:

- *Validation rules* – These rules specify the restrictions applied to data values. For example, the value for a particular data item may be restricted to a certain data type (e.g. an integer) and limited to a certain range (e.g. > 0).
- *Workflow rules* – These rules are concerned with flow control. For example, to whom can the workflow object be sent? In what order (of recipients) must the object traverse? The flow may be sequential, conditional (branching condition), iterative (loop condition), etc.
- *Access control rules* – These rules describe the access permissions assigned to a data item. The types of access control permissions supported are discussed in Section 3.7.
- *Requirements rules* – These rules express the conditions that the workflow object must satisfy

before a verifying workflow participant certifies the workflow object. Such rules could include expecting data values (e.g. data item x must be null), expecting protected data items (e.g. x must be *read only* for role G), separation of duty, etc.

The *validation* and *workflow* rules are specified in the schema, whereas the *access control* and *requirements* rules are specified in the policies.

The schema is formed of two parts – the *structure description* (containing the validation rules) and the *workflow description* (containing the workflow rules). The structure description, inspired by the W3C’s XML Schema model [92], also governs the structure of the object data component.

The policies component is partly based on the XACML policy model [20], with extensions added to support requirements rules. The constraints defined in the policies component are expressed by the workflow participants (in particular, data owners) and verifiers. On the other hand, the constraints defined in the schema component are expressed by the issuing authority.

The *template* component describes the presentation format for the workflow object. To support automated processing, the object data and object metadata components must be machine interpretable. But such a format cannot be easily read by human users. To make the workflow object user-friendly for workflow participants, we define a template for it. Using this template, we can automatically generate a human readable view of the workflow object. This view serves as the interface between the workflow participant and the workflow object.

4.2 Specifications

This section defines the specifications for each of the five workflow object components, using Extended Backus-Naur Form (EBNF) [19] to express how these components are formed. The typesetting used in the EBNF expressions have the following semantics:

- Elements in sans-serif font indicate variable names.
- Elements in *italics* indicate constants that are defined elsewhere (dependent on tools).
- Elements in **bold** indicate constants.

4.2.1 Object Data Component

The object data organises data items in a tree structure. It contains a single *root element* data item from which all other data items within this component can be accessed. A data item exists as either a *name-value* pair or it contains child data items. This component is expressed as follows:

```
object-data ::= data-item ;
data-item  ::= ( item-name , item-value ) | data-item+ ;
item-name  ::= alpha-char+ ;
item-value ::= char* ;
alpha-char ::= alphanumeric-characters ;
char       ::= characters ;
```

4.2.2 Object Metadata Component

The object metadata component comprises of four parts: (1) a reference to the corresponding object data, (2) the workflow identifier for this workflow object, (3) workflow history records, and (4) protected data item records.

Workflow history records contain the user's ID, the role being enacted by the user, the record type (either acquire or release) and a signature from the identity provider validating the record. Protected data item records contain a reference to the protected data item, the list of authorised subjects and other protection information (e.g. signatures) required for protecting the data item.

The object metadata component is defined as follows:

```
object-metadata ::= ( object-data-ref , workflow-id , workflow-history-record* ,
                    protected-data-item-record* ) ;
object-data-ref ::= reference-type ;
reference-type  ::= references ;
workflow-id     ::= workflow-identifiers ;

workflow-history-record ::= ( user-id , role-id , history-record-type , signature ) ;
user-id                ::= user-identifiers ;
```

```
role-id           ::= role-identifiers ;
history-record-type ::= ACQ | REL ;
signature         ::= digital-signatures ;

protected-data-item-record ::= ( data-item-ref , subjects , protection-info ) ;
data-item-ref           ::= reference-type ;
subjects                ::= role-id+ ;
protection-info         ::= ( access-type , signature* , attribute* ) ;
access-type             ::= read | write | append ;
attribute               ::= char+ ;
char                    ::= characters ;
```

4.2.3 Schema Component

The schema consists of a structure description and a workflow description. As discussed in Section 4.1, the structure description defines the data item hierarchy of the object data component. It contains type definitions for the data items as well as any data constraints that may apply to item values in the object data.

The workflow description defines the workflow and any constraints (workflow rules) that may exist. The workflow is specified in terms of workflow activities (Section 2.1.1) and workflow patterns (Section 2.1.3) as the precise workflow specification is tool dependent.

Following is the schema component expressed in EBNF:

```
schema ::= ( structure-description , workflow-description ) ;

structure-description ::= data-item-def+ ;
data-item-def         ::= ( item-name , type-def , data-constraint* ) ;
item-name             ::= alpha-char+ ;
alpha-char            ::= alphanumeric-characters ;
type-def              ::= type-definitions ;
```

```
data-constraint ::= data-constraints ;

workflow-description ::= ( workflow , workflow-constraint* ) ;
workflow ::= workflow-activity | ( workflow , workflow-pattern ) ;
workflow-activity ::= workflow-activities ;
workflow-pattern ::= workflow-patterns ;
workflow-constraint ::= workflow-constraints ;
```

4.2.4 Policies Component

The specification for the policies component is largely influenced by the XACML policy model [20]. Each policy specifies a target, one or more rules and an optional conflict resolution strategy. A target includes one or more subjects (i.e. to whom the policy is applicable), one or more resources (i.e. the targeted data items) and the applicable actions. Each rule also contains a target, as well as an effect and an optional condition. Effect values for access control rules are *permit* or *deny*, whereas effect values for requirements rules are *accept* or *reject*. The *indeterminate* effect value applies to both types of rules and represents an error case. We specify this component in EBNF as follows:

```
policy-set ::= ( target , policy+ , conflict-resolution? ) ;
policy ::= ( target , rule+ , conflict-resolution? ) ;

target ::= ( subject+ , resource+ , action* ) ;
subject ::= role-identifiers ;
resource ::= references ;
action ::= read | write | append | any ;

rule ::= ( target , effect , condition? ) ;
effect ::= permit | deny | accept | reject | indeterminate ;
condition ::= boolean-expressions ;
```


conflict-resolution ::= **first-applicable** | **permit-overrides** | **deny-overrides** |
accept-overrides | **reject-overrides** ;

4.2.5 Template Component

The template component specifies the presentation format for the workflow object. It is not limited to a particular set of file formats (e.g. HTML, PDF, etc.). Instead, we visualise the template as a *form* and define it in terms of form elements (e.g. labels, text boxes, etc.) linked to data items. This is specified below:

template-form ::= form-element* ;
form-element ::= *form-elements* ;

4.3 Summary

This chapter has presented the workflow object model that has been proposed in conjunction with the architecture model (discussed in Chapter 3). The workflow object model defines how finer-granularity access control is applied to sensitive data within the workflow object. In addition, this model specifies how constraints are applied to the workflow object.

The next chapter discusses a prototype implementation design for the proposed models. This prototype will be subsequently used for conducting case studies.

Chapter 5

Prototype Implementation

The prototype implementation aims to show the practical feasibility of the architecture model. It provides the foundation for developing case studies to examine applications for this research.

The prototype is not a complete implementation of the architecture model. We limit its scope to the main contributions of our work, viz. fine-grained access control enforcement in foreign security domains and role-based workflows. Our intention is not to develop a fully fledged product but to provide experimental support for the case studies.

Similar implementations concentrate on access control for XML documents [4, 15, 16]. Although these examples are based on different architecture models and use different implementation tools, the same principles exist: access control enforcement using cryptography, policy specification for access control and verification, key management, etc. We also use XML documents in this prototype to implement the workflow object.

In addition to explaining the design aspects of the implementation, this chapter discusses the design decisions and experiences encountered whilst implementing the prototype.

5.1 Overview

The prototype consists of three components: *Workflow Engine*, *Identity Provider* and *Document Processing Environment*. Figure 5.1 shows how the user interacts with each component. The subsequent sections of this chapter discuss each of the prototype components in further detail.

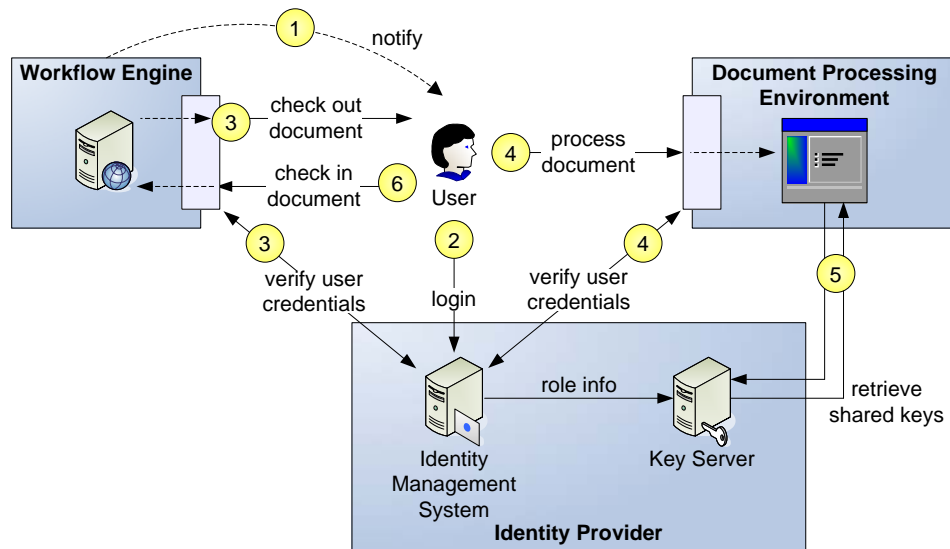


Figure 5.1: Overview of the prototype implementation setup.

The workflow engine component implements document flow. As the workflow progresses, the document eventually becomes ready for the user to process. At this point, the user retrieves the document from the workflow system and processes it within the document processing environment. So that the processing application can interact with the identity provider (to retrieve shared keys), the user must first authenticate himself. This is performed via the identity management system at the identity provider. The key server also interacts with this identity management system to retrieve the user's role details since shared keys are dependent on roles (not user identities). When the user has finished processing the document, it is sent back to the workflow system for the next workflow participant to retrieve.

In regards to how the workflow is set up, this requires agreement between all participating organisations. How this agreement is reached is out of the scope of this thesis. Responsibility for constructing the workflow document lies with the organisation acting as the Issuing Authority. As for user management, each organisation handles this separately, and this is exposed through their Identity Provider. It is assumed that trust relationships exist between the Identity Providers of the participating organisations.

5.2 Workflow Engine

We used a document management system (with workflow capabilities) to implement the workflow part of the architecture model. This provided three important functions: *workflow specification*, *enforcement of workflow constraints* and a *simulated communications medium*. In the architecture model, the method for transferring the workflow object between recipients remains unspecified. This decision is effectively left to the developer.

KnowledgeTree [39] is the tool that we selected as our document management system. This is a commercial, open source product that provides a web-based interface and offers document collaboration features such as document alerts and, more importantly, workflows. We decided on this tool for its workflow design capability and role-based support. The workflow design component allows developers to apply access control restrictions (assigned to roles) for each workflow state. Given that the architecture model adopts a role-based workflow approach, this is an especially useful feature.

At first, we considered using a workflow management system for implementing the workflow. We experimented with JBoss jBPM [37] but found that although it provided a rapid development environment for designing workflows, it lacked document flow support (such functionality required customised coding). Furthermore, its user management capabilities and documentation were less extensive to those of KnowledgeTree. In fact, JBoss jBPM only supported user-based workflows.

So that users can access the KnowledgeTree system, we configure it with user accounts and their corresponding role memberships. To an extent, KnowledgeTree acts as an *identity provider* (as it provides user authentication). However, this authentication is limited solely to acquiring the document. For key management (e.g. issuing shared keys), we must use a separate system (discussed later in this chapter). This separate system actually represents the real identity provider. We could have instructed the KnowledgeTree system to retrieve user information stored, for example, in an LDAP directory located at the real identity provider. But as this is only a proof-of-concept prototype, it suffices to use KnowledgeTree's own user management capabilities.

The workflow engine setup is illustrated in Figure 5.2. Whenever a new document is added to the system, it is assigned to a workflow. By workflow, this refers to a workflow description that is already defined in the system. Depending on the current state of the workflow, the document will only

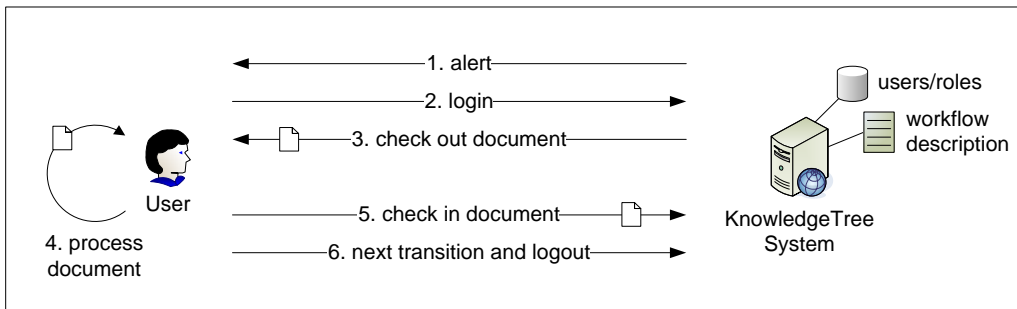


Figure 5.2: Workflow simulation setup for prototype.

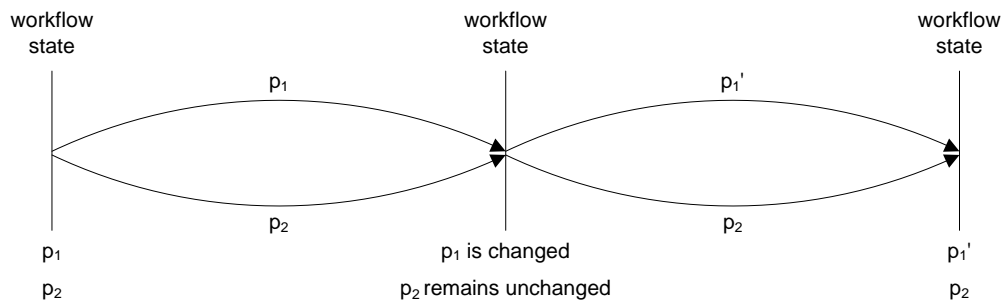


Figure 5.3: Parallel transitions.

be accessible to certain users (i.e. to those users who are authorised to acquire the document in its current state).

As the workflow progresses and the document becomes ready for the next recipient, the system sends an alert to the user who will act as the next recipient. This user then logs into the KnowledgeTree system and checks out the document (to prevent any other users from modifying it). Once the user has finished processing the document, they check it back into the system and perform the next workflow transition. After a transition is performed, the document moves to a new workflow state. Depending on the workflow description, the user may no longer have access to the document. Performing a transition simulates moving the document from one recipient to another.

One could argue that since KnowledgeTree itself provides access control restrictions, each protected data item in the workflow object could exist as a separate document and be protected using KnowledgeTree's access controls (hence eliminating the need for encryption keys). In other words, we could have many smaller documents forming a single logical document. The challenge with this would be coordinating the flow of all of the smaller documents.

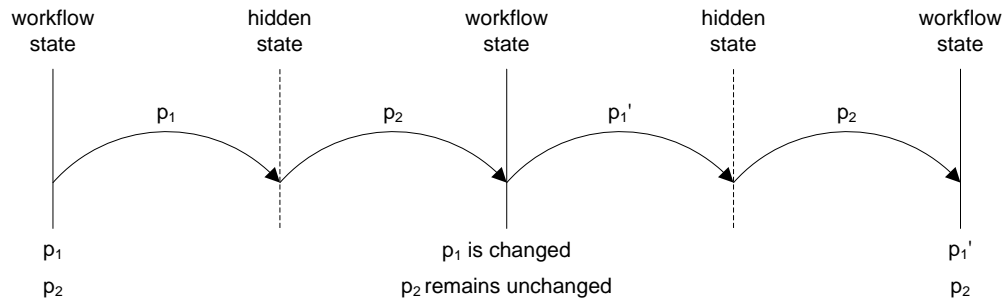


Figure 5.4: Intermediary transitions.

Figures 5.3 and 5.4 show two possible approaches for implementing this scenario. In each of these figures, document d is divided into smaller document parts p_1 and p_2 . Each document part exists as a separate document.

One option is where the workflow has concurrent processes. In other words, we have parallel flows of execution. Another option is where the workflow has only a single process but includes “hidden states”, as well as actual states, so that the previous parallel flows are remodelled into a series of intermediary transitions. Although this is less efficient, it doesn’t experience synchronisation issues associated with parallel flows (e.g. when merging flows).

As of the time this prototype was under construction, none of the above scenarios could be implemented with KnowledgeTree. Each workflow instantiation in KnowledgeTree can only have one document and parallel workflows are unsupported. Furthermore, synchronisation for simultaneously executing workflows is also unsupported. Despite alternative tools possibly implementing these scenarios, a fundamental aim of this research is embedding access control measures within the workflow object as opposed to relying on various systems to enforce this.

5.3 Identity Provider

As introduced in Section 5.1, the identity provider serves two functionalities:

1. An identity management allowing users to authenticate themselves. User authentication is required to access the document processing environment.
2. A key server for issuing shared keys referenced in the document.

The implementation of each of the above functionalities is discussed in the following subsections.

5.3.1 Identity Management

We selected Shibboleth [65] as our identity management system. Shibboleth is an open, standards-based solution for securely exchanging information about users. A key feature of this tool is *access control based on attributes*.

Consider the situation where a user belonging to *home domain* wants to access a protected resource at *target domain*. Rather than the target domain trying to authenticate the user, it could instead request information about the user (e.g. member of an institution or particular class) from the user's home domain. Based on this information, an authorisation decision can be made by the target domain on whether or not the user has permission to access the resource. A trust federation that includes both domains must exist for this scenario to work (as authentication is delegated to the user's home domain). For the prototype, all users and roles are implemented within the one domain, hence we do not establish a trust federation. Effectively, the prototype uses a centralised implementation.

Shibboleth (version 1.3 and earlier) does not provide built-in authentication support. To authenticate users, we used a web-based system named Central Authentication Service (CAS) [10]. CAS is a single sign on (SSO) service that can handle user authentication against different backends (e.g. LDAP, SQL databases, NIS, Kerberos, etc.). We configured Shibboleth with a CAS filter to enable CAS to perform user authentication on behalf of Shibboleth. We used LDAP as the backend for our CAS implementation.

Using a federated identity management system like Shibboleth (as opposed to simply using an authentication system) eliminates the need for individual content providers to maintain user information such as usernames and passwords. Shibboleth recognises two main types of entities: *identity providers* and *service providers*. Identity providers authenticate users and release user information to service providers. Service providers consume this information to decide whether to grant access to its secure content. This is how Shibboleth implements access control based on user attributes. For this prototype, the user attribute of interest is the user's role.

The identity management setup for the prototype is illustrated in Figure 5.5. There are two parts:

1. The identity provider part is hosted on the identity provider.

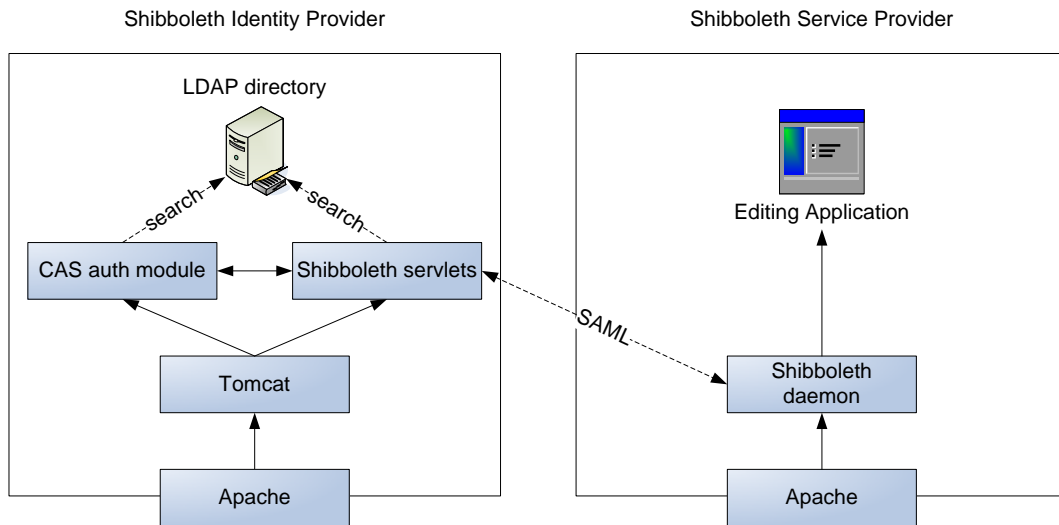


Figure 5.5: Identity management system setup.

2. The service provider part runs in the document processing environment.

The Shibboleth identity provider runs an Apache web server listening on port 443. Requests to this port are redirected to the Apache Tomcat server that hosts the Shibboleth servlets and the CAS authentication module. User information is stored in a LDAP directory that is also located on the identity provider.

On the service provider, a Shibboleth daemon runs in the background. When a connection is made to the service provider, the daemon intercepts this connection to check whether the user is already authenticated with Shibboleth. If not, the connection is redirected to the identity provider (more precisely, the CAS login page) where the user inputs their username and password. If the login is successful (i.e. a matching user record is found in the LDAP directory), the Shibboleth identity provider servlets exchange SAML [61] tokens with the Shibboleth daemon on the service provider. These tokens reveal the role membership(s) of the authenticated user. If the user cannot be authenticated, the daemon denies access to the document processing environment.

5.3.2 Key Management

Key management in the prototype focuses on issuing shared keys to authorised users. To accomplish this, we created a Java Servlet implementing the key exchange algorithm discussed in Section 3.6.

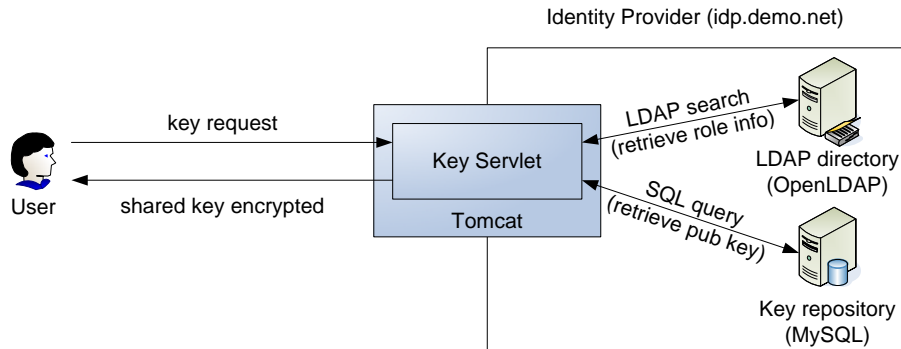


Figure 5.6: Key management setup for prototype.

This servlet is hosted by the identity provider on the same Tomcat web server as Shibboleth. This conforms to the architecture model specifications, which states that shared keys are issued to users by the identity provider.

The prototype's key management setup is displayed in Figure 5.6. Firstly, the user asks the identity provider for a shared key by sending a request message to the servlet. This key request is formed as follows:

$$key\ request = \{u, \{subject, id_{WF}\}_{sig_u}\}$$

where u = the userid of the user sending the request

$subject$ = those roles between whom the shared key is intended for

id_{WF} = the workflow ID as defined in Section 3.5

Next, the servlet executes the following algorithm to process the request and issue the shared key:

1. Retrieve the user's public key from the key repository to verify the signature contained in the request message. If the signature is invalid, reject the key request.
2. Check whether the user is authorised to receive the requested shared key. If at least one of the user's roles is listed in the subject of the key request, then the user is authorised to receive the shared key. This step involves retrieving the user's roles from the LDAP directory. Reject the key request if the user is unauthorised.
3. Create the shared key by calculating the hash of $(subject, timestamp, secret)$. This is consistent

with the formula defined in Section 3.6. The values for *timestamp* and *secret* are extracted from the workflow ID.

4. Encrypt the shared key with the user's public key and send to user.

During the processing, the servlet connects to an LDAP directory containing user/role information and to a key repository containing the public key for each user. It is a requirement of the architecture model that each user has their own public/private keypair. We implemented the key repository as a MySQL [48] database system. Although we could have stored public keys in the LDAP directory, we decided against this as we did not want to modify the LDAP record structure and potentially affect the Shibboleth setup.

For generating the public/private keypairs, we used the Java Cryptography Architecture API [36]. Alternatively, we could have used GnuPG [30] or certificates issued by the identity provider, but we selected the API so that we could easily use its other cryptographic-related functions (e.g. encryption, decryption, signature verification, etc.) and avoid potential compatibility issues.

5.4 Document Processing Environment

The document processing environment resides at the user's end. Its purpose is to take the document (specifically, the data and metadata components), and display it in a user-friendly view for the user to read and edit. It also automates key retrieval operations for when the user protects data items.

Figure 5.7 shows the implementation setup for the document processing environment. The following steps explain how this setup works:

1. The document data and document metadata are "loaded" into the document processing environment by the user. These components are implemented as XML files. The files form the workflow document that is checked out from the workflow engine.
2. The user views and edits the contents of these XML files via the document template. We have implemented this template as an HTML file, allowing the user to process the workflow document via a standard web browser. This decision provides consistency with other parts

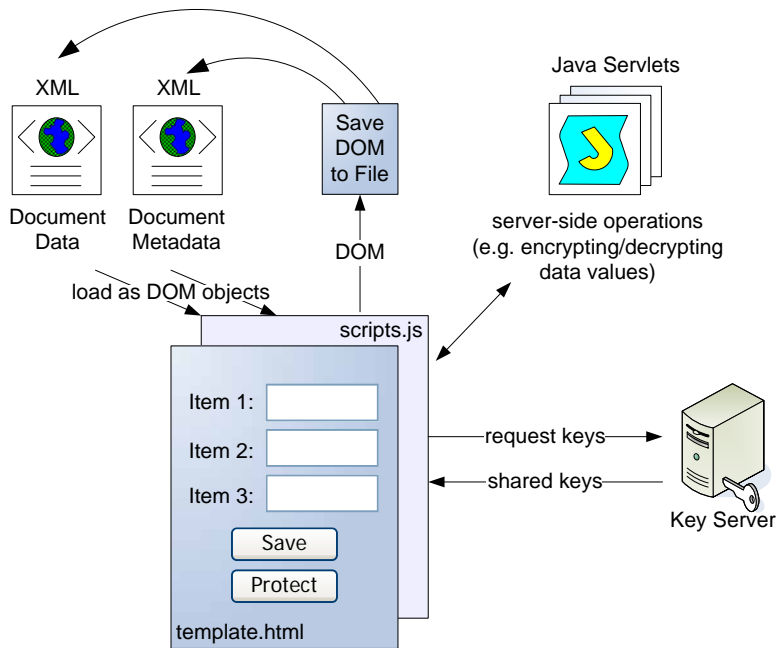


Figure 5.7: Document processing environment setup.

of the prototype implementation (e.g. KnowledgeTree and Shibboleth) which also use a web interface.

When the user opens the HTML file in the web browser, a scripts file linked to the HTML file is executed. The scripts, coded in JavaScript, creates DOM objects for the XML files, to load the document data and document metadata into memory. The scripts then proceed with scanning through the metadata to identify protected data items that the user is authorised to access, and attempts to retrieve the shared keys corresponding to these data items. If a protected data item can be successfully unlocked, its value is displayed on the HTML page. Otherwise, a concealed value is displayed for the data item.

As we rely on the Java Cryptography Architecture API to unlock data items, we cannot use JavaScript. Instead, we implement a Java Servlet, executing at the user's end, to perform the cryptography related functions necessary to process protected data items. The servlet is invoked from the scripts file.

3. Once the metadata has been parsed, all data item values that are accessible to the user are

displayed on the HTML page. Our template design appears much like a web form, such that each data item value is displayed in a text box and events (e.g. protect item, save changes, etc.) are triggered via command buttons. If a data item is protected as read only, the text box is disabled so that no changes can be made. This is also the case for text boxes containing a concealed value. For unprotected data items and data items with read write permissions, the text box is enabled so that the value can be edited.

To protect a data item, the user clicks the “protect” command button assigned to that data item. This brings up some properties that the user must set. These properties include the type of protection - that is, *read only* or *read write* – and the key subject (the roles authorised to access the data item). After setting these properties, a new metadata record is created and inserted into the document metadata DOM object. Again, we use Java Servlets to create this new record as cryptography related functions are invoked during its creation. We must also request a shared key from the key server for protecting the data item and establishing the metadata record.

4. Before finishing with the document and submitting it back to the workflow engine, any changes need to be saved. To save the document, the user clicks the “save” command button. This invokes a script that scans through all of the data items, looking for changes to protected data items. If a changed value is detected, the old value is overwritten with the new value, and the data item is reprotected. Finally, the DOM objects are saved to disk, overwriting the original XML files used to initialise the DOM objects. The XML files are now ready to be returned to the workflow engine.

As mentioned, we require Java Servlets to perform the server-side operations that cannot be handled by JavaScript. These servlets are hosted on an Apache Tomcat web server. The directory containing the template is guarded by a Shibboleth daemon, running on an Apache web server (see Figure 5.5). When the user opens the template page in the web browser, the connection is intercepted by the Shibboleth daemon, which checks to ensure that the user is authenticated and authorised to open the page. If the user is not authenticated, the daemon redirects the connection to a login page at the Shibboleth identity provider. Once authenticated, the daemon allows access to the template page.

We use an ASP script to write DOM objects to file. Although this uses a different server-side

language and requires the Microsoft IIS web server, the script was already available to us, hence saving development time.

5.5 Summary

The prototype implementation presented in this chapter contains three components: a *workflow engine*, an *identity provider* and the *document processing environment*. Where possible, we have attempted to “plug-in” existing tools to perform the necessary functionalities. This prototype is by no means an optimal implementation of the architecture and workflow object models. It is simply an experimental system to support the case studies discussed in the next chapter. As such, we keep the design flexible so that it can be easily applied to multiple scenarios without requiring major reconfiguration.

Chapter 6

Case Studies

This chapter presents the results of the case studies that were performed to demonstrate potential applications for this research. The case studies aim to show the practical feasibility of the models. We look at three different scenarios, applicable to e-business, e-health and e-government respectively. Each subsequent case study expands on the previously performed case study.

6.1 Case Study 1 – Loan Application

This case study is based on a banking e-business scenario taken from the SERENITY (System Engineering for Security and Dependability) Integrated Project [8]. The scenario looks at a loan application submission to a money lending institution. In practice, such financial-related scenarios are challenging, not only because of their need to comply with strict regulations, but also because of the various security, privacy and trust issues involved. Despite the complex barriers to real-world adoption, this case study aims to give an indication of the potential benefits that secure document circulation can offer to e-business.

The workflow is illustrated in Figure 6.1. It starts with the customer submitting a loan application form to the money lending institution. A pre-processing clerk (either from the lending institution or outsourced) performs an identification check on the customer. This requires the customer to present proof of ID to the pre-processing clerk. After confirming the customer's ID, the application is passed over to the credit bureau for a credit worthiness check. The results of this check are sent back to

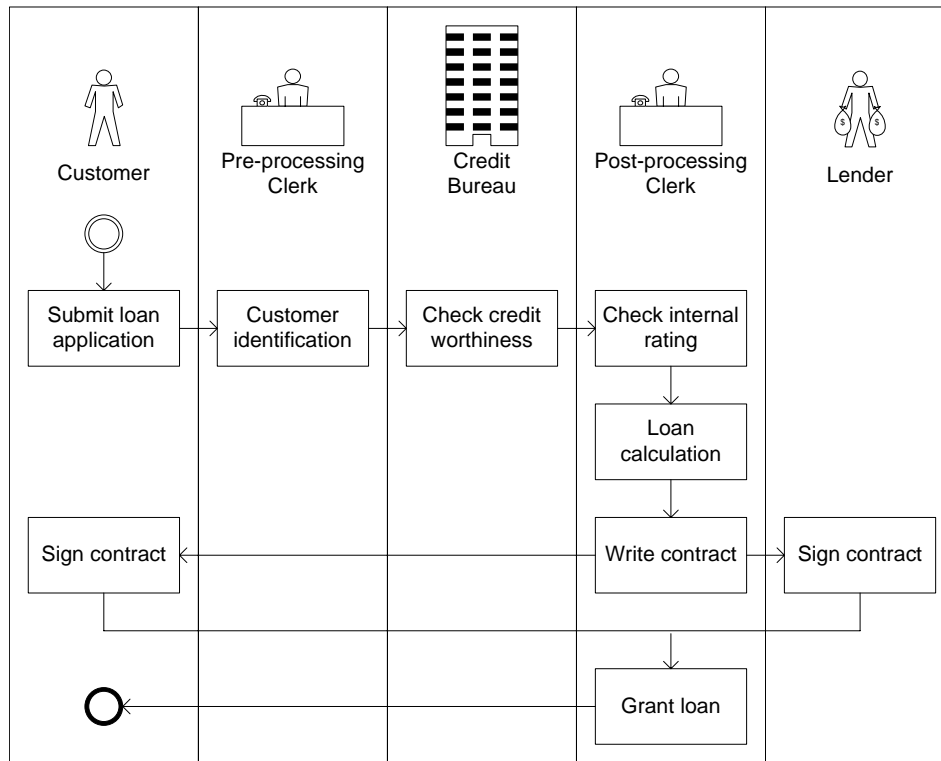


Figure 6.1: Loan application workflow.

the lending institution, where they are used by a post-processing clerk to determine the customer’s internal rating. This rating represents the lending institution’s own risk assessment on the customer. If satisfactory, a loan amount is calculated and the contract is arranged. The final stage involves the customer and the lender signing the contract, followed by the loan being issued to the customer.

The objective of this case study is to determine whether the implementation approach (discussed in Chapter 5) works, with an emphasis on the access control aspects of the architecture model.

6.1.1 Process

We now look at the workflow step-by-step, and how we can deploy it using the prototype discussed in Chapter 5.

1. The first step requires the customer to fill out the loan application form and submit it to the lending institution. We assume that the lending institution allows customers to complete the application form via its website. The customer connects to the website, completes the online

application form and submits it.

The online application form serves as an interface to the workflow document. Data that the customer enters into the online application form is transferred to the workflow document (which is created when the customer submits the online form). The workflow document is actually a machine-interpretable representation of the loan application form. Each workflow document instantiation represents a new workflow instance becoming invoked.

After the workflow document has been initialised with the data from the online form, the document is checked into the lending institution's Document Management System (DMS). The DMS assigns the document to the *pre-processing clerk* role for processing.

2. A pre-processing clerk logs into the DMS and retrieves the workflow document for processing. During this step, the clerk performs a customer identification check. This clerk may be from the lending institution or outsourced from a different company. The customer meets the pre-processing clerk in person to present some ID to prove their identity as per the legislative requirements.

As the clerk retrieves the workflow document, it is checked out of the DMS. The clerk has a front end software application (document processing environment) to which the document is loaded into for viewing and editing. When convinced of the authenticity of the ID presented by the customer, the clerk records the details of the ID into the document (via the front end application).

Any confidential customer ID details stored in the document need to be protected so that only authorised roles can view this information. The clerk protects these details using shared keys requested from the lending institution's key server. Information such as the data items protected, the shared keys used and the authorised roles (e.g. credit bureau employee, lending manager, etc.) are all saved as metadata in the workflow document's metadata component.

After processing the workflow document, the pre-processing clerk returns it to the DMS. The clerk directs the DMS to alert the next workflow participant (i.e. the credit bureau) that the document is ready for them to process.

3. An authorised employee from the credit bureau logs into the DMS and retrieves the workflow document to perform a credit worthiness check on the customer.

The credit bureau has their own specialised front end application for viewing and editing the workflow document. The employee performs the credit worthiness check using customer information stored in the document. The results from the check are written to the document and protected similarly to the previous step.

When finished, the credit bureau employee checks the document back into the DMS. The document is now assigned to the *post-processing clerk* role for further processing.

4. This step requires a post-processing clerk (from the lending institution) to check the customer's internal rating. This rating is a reflection of the lending institution's own risk assessment on the customer, based on information declared by the customer (e.g. annual salary, weekly expenses, etc.) and the outcome of the credit worthiness check.

As in the previous steps, the clerk logs into the DMS, retrieves the workflow document and loads it into the front end software application for handling the internal rating assessment. The results of this assessment are recorded within the document, and a decision on whether to approve or deny the loan is made based on these results. The results may also be protected so that only the lending manager can edit them.

The remaining tasks in the workflow concern the creation and signing of the loan contract. Although the workflow now proceeds with a new document (i.e. the contract), it uses the workflow document as its basis.

6.1.2 Document Design

We illustrate the design of the workflow document using XML schemas. As mentioned, this case study recognises two components of the workflow document - i.e. the *document data* and the *document metadata*.

Figure 6.2 shows a top-level graphical view of the schema for the document data component. The structural design of this component is representative of an actual "paper-based" loan application form.

The various sections defined in this design can be organised into three categories:

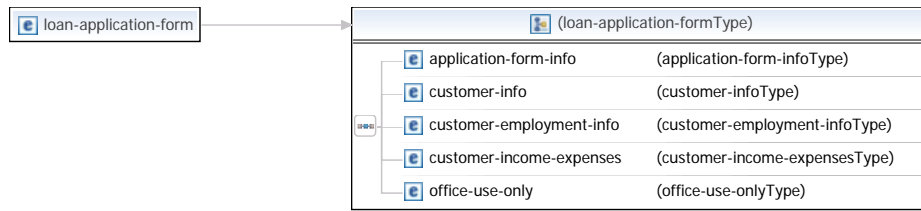


Figure 6.2: Top-level view of document data component.

1. *application-form-info* contains data that is written to the form when the workflow document is initialised. Such data includes the application form number, the date when the loan application was made and the loan amount being applied for.
2. *customer-info*, *customer-employment-info* and *customer-income-expenses* contain data that is provided by the customer. The customer provides this information during the first step of the workflow.
3. *office-use-only* contains data that is written to the form during processing of the loan application. For the case study, we concentrate mostly on this section.

Figure 6.3 examines the structure of the office-use-only section in further detail. As shown, this section is organised into three subsections - one for each check performed on the customer during the processing of the application form. Each of these subsections becomes a protected data item once its corresponding check has been performed.

For the first check (i.e. the customer identification check), the pre-processing clerk records details of the ID documents presented by the customer. In addition to these details, the clerk authorises and dates the form, before protecting this information.

The subsequent checks (i.e. the credit worthiness check and the customer risk analysis) follow a similar process. The workflow participant completes the respective subsection and protects it. Details on the level of protection applied to these subsections is discussed in Section 6.1.4.

The design for the document metadata component is displayed in Figure 6.4. This design is somewhat generic, given its close resemblance to the workflow object metadata design discussed in Chapter 4. Hence it need not be solely confined to this loan application case study.

The document metadata is composed of three parts:

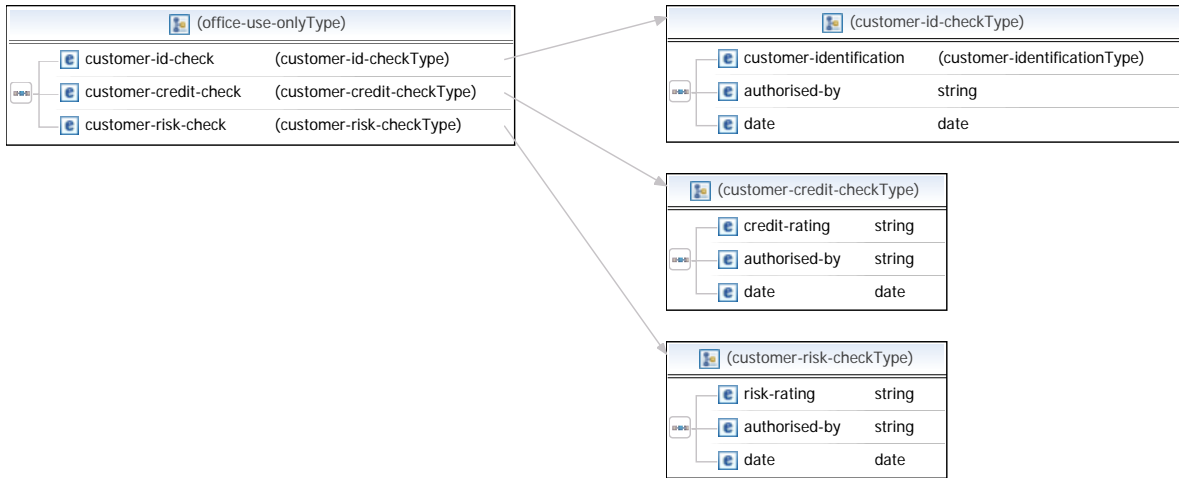


Figure 6.3: Document data component section for processing.

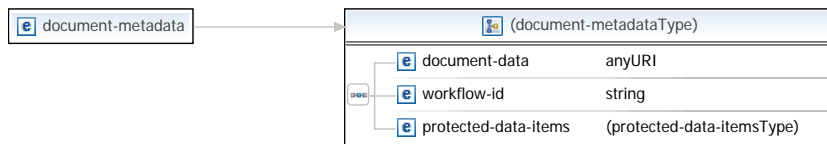


Figure 6.4: Top-level view of document metadata component.

1. *document-data* contains the location of the document data linked to this metadata.
2. *workflow-id* contains the value of the workflow ID for this workflow document.
3. *protected-data-items* contains a collection of protected data item records.

Each protected data item in the document data has a record in the protected-data-items section. The formation of the protected data item record is shown in Figure 6.5. The *ref* element acts as a pointer to the protected data item. The *protection-type* element states the access type applied to

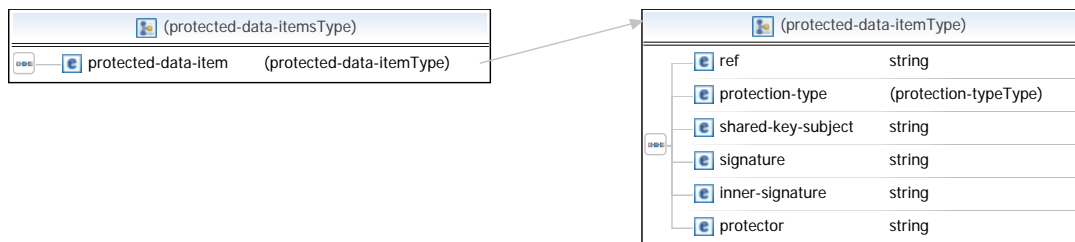


Figure 6.5: Protected data items section in document metadata component.

the protected data item. For this case study, we support *read only* (ro) and *read write* (rw). The *shared-key-subject* element is used to retrieve the shared key for unlocking the protected data item. The remaining elements relate to the encryption algorithms defined in Section 3.7. The elements *inner-signature* and *protector* are only required for read only access.

6.1.3 Users and Roles

There are four roles that are prominent in the business process discussed in Section 6.1.1: *customer*, *pre-processing clerk*, *credit bureau employee* and *post-processing clerk*.

Given that the focus for this case study is on document processing (in particular, applying access control restrictions to data items), we concentrate on the processing steps and start the implementation from step 2. Hence we assume that the first step (where the customer fills out and submits the loan application form) is already completed and the workflow document is awaiting processing by the pre-processing clerk. For this reason, we do not require the customer role for this case study.

We define a separate user identity to enact each of the three remaining roles. The user to role mappings are implemented as follows:

- user *alpha* acts as the pre-processing clerk
- user *beta* acts as the credit bureau employee
- user *gamma* acts as the post-processing clerk

Each user has their own public/private keypair as well as a login account with the Shibboleth identity provider and the KnowledgeTree DMS. User to role mappings are defined in an LDAP directory (located at the identity provider), and this information is retrieved by Shibboleth when authenticating a new session. An extract of this user information from the LDAP directory appears as:

```
dn: uid=alpha,dc=demo,dc=net
objectClass: account
objectClass: simpleSecurityObject
uid: alpha
userPassword:: YWxwaGFwdw==
```

description: pre-processing_clerk

The description attribute defines the roles enacted by the user. Although not shown here, if the user has multiple roles, the role names in the description attribute are delimited with a colon character (e.g. description: programmer:tester).

6.1.4 Implementation

We now discuss how we executed this case study using the prototype implementation. Before instantiating the workflow, we must first set up the following parts to the implementation:

- User account setup
- Workflow description setup
- Document template setup

As mentioned in Section 6.1.3, user accounts are represented as LDAP entries at the identity provider end. User authentication is also required for the workflow engine. To ease the implementational complexity, we use KnowledgeTree’s own user authentication system rather than attempting to integrate the KnowledgeTree DMS into our Shibboleth “single sign-on” setup. Although this suffices for a prototype, all user authentication would be handled solely by the identity provider in an actual implementation (to avoid the need to maintain multiple user information sources).

For the workflow description, KnowledgeTree’s workflow component allows us to specify the workflow path completely within the KnowledgeTree system. This includes specifying the various workflow states, all possible transitions between states, and the user permission sets at each workflow state. Once the workflow has been defined, we can upload the (newly instantiated) workflow document into KnowledgeTree and assign the workflow description to the document. From this point on, KnowledgeTree enforces the workflow conditions applied to the document.

The document template is tightly coupled to the document data design (see Section 6.1.2). For this reason, we concentrated on a customised template design purely for this case study instead of attempting to generate a standard template design. We did however manage to identify sections in

our template design which can be extracted and reused in future case studies. Reusable template fragments are particularly evident in the template's scripts file, where much of the document processing is defined. For example, the functions for loading the workflow document, saving the workflow document and protecting a data item can all be coded such that they are abstract from the underlying document data and document template designs.

6.1.4.1 Executing the workflow

The workflow execution involves three machines.

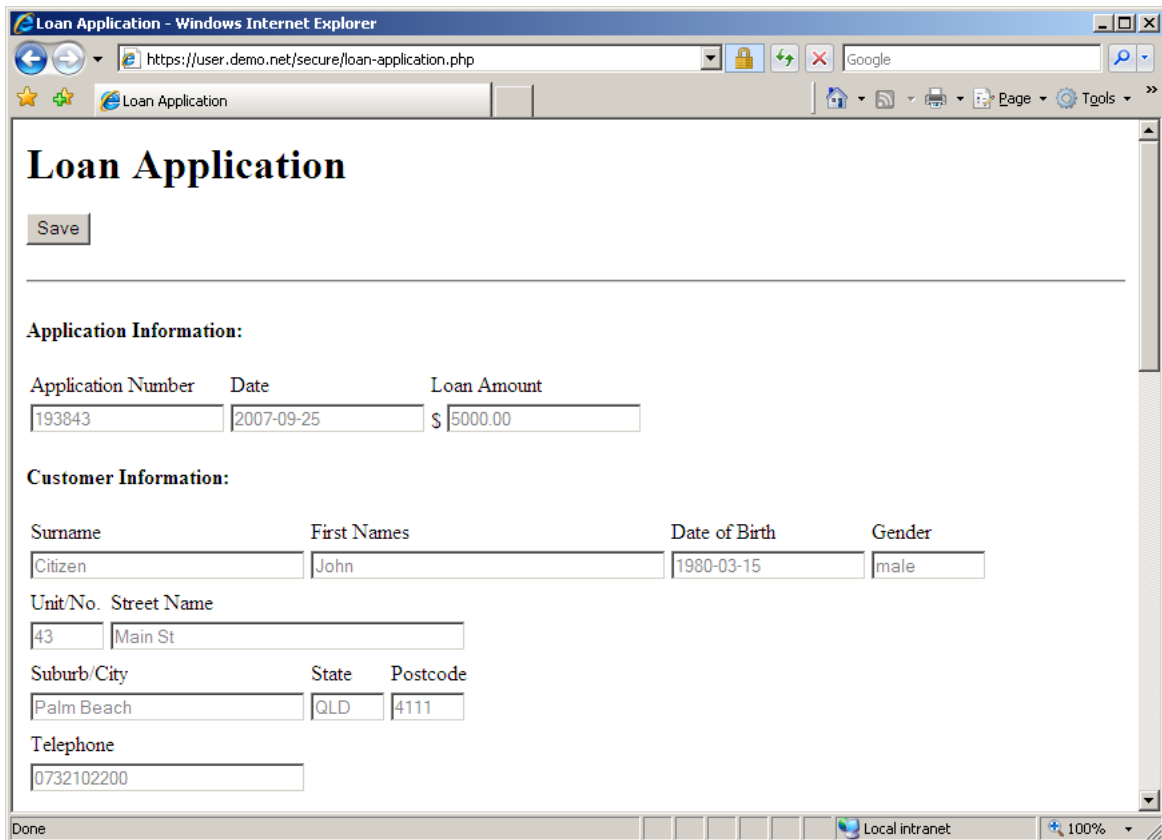
1. `wfms.demo.net` hosts the KnowledgeTree DMS. This machine also hosts a mail server for the purposes of email notifications.
2. `user.demo.net` represents the end user machine. The document processing environment operates on this machine. Although in reality each user would have a separate machine, for demonstration purposes, it suffices for each user to share the one machine.
3. `idp.demo.net` hosts the identity provider. This machine operates the Shibboleth identity provider component and the key management server.

The workflow starts with the first user (i.e. *alpha*) connecting to KnowledgeTree to check-out and download the workflow document to a designated directory on the end user machine. The workflow document is implemented as a compressed (zipped) file containing two XML files (representing the document data and document metadata). Once downloaded, the workflow document is unzipped and the template file is opened in the web browser. These steps (from connecting to KnowledgeTree to unzipping the workflow document) are performed manually by the user. Ideally, these steps would be automated and performed when the user opens the template file. But since this is not a priority of the prototype, it remains as a manual process.

The template file is actually hosted on a locally installed web server at the address:

```
https://user.demo.net/secure/loan-application.php
```

On opening this file, a redirection to the Shibboleth login page is made. This automatically occurs since the end user machine is configured for the Shibboleth daemon to intercept any connections to



The screenshot shows a web browser window titled "Loan Application - Windows Internet Explorer". The address bar displays "https://user.demo.net/secure/loan-application.php". The page content includes a "Save" button, a section titled "Application Information:" with fields for "Application Number" (193843), "Date" (2007-09-25), and "Loan Amount" (\$5000.00). Below this is a section titled "Customer Information:" with fields for "Surname" (Citizen), "First Names" (John), "Date of Birth" (1980-03-15), and "Gender" (male). Further down are fields for "Unit/No." (43), "Street Name" (Main St), "Suburb/City" (Palm Beach), "State" (QLD), "Postcode" (4111), and "Telephone" (0732102200). The browser status bar at the bottom shows "Done" and "Local intranet".

Figure 6.6: Screenshot of loan application form.

files contained in the `secure` directory. The user inputs their username and password, and a new authenticated session is established. The user is now redirected back to the template file.

Section 5.4 states that the template is implemented as an HTML file, but in the above web address, it is listed as a PHP file. This is because we insert PHP code into the HTML page to extract user information from the Shibboleth session (specifically, the user's name and role memberships). These attributes are subsequently used by the template's JavaScript functions to perform the various document processing operations (e.g. request shared keys, etc.).

Screenshots of the template are displayed in Figures 6.6 and 6.7. The latter screenshot shows data items that can be (or have already been) protected. In this case, the *customer identification check* data item is concealed, indicating that the user viewing the template does not have permission to obtain the shared key for this data item.

The screenshot shows a web browser window titled "Loan Application - Windows Internet Explorer". The address bar displays "https://user.demo.net/secure/loan-application.php". The page content is organized into three sections, each with a "Protect" button:

- Customer Identification Check:** A table with five columns: ID Type, ID Issued By, ID Number, ID Date of Issue, and ID Place of Issue. Each cell contains a text input field with "*****" as a placeholder. Below the table are two more input fields labeled "Authorised By" and "Date", followed by a "Protect" button.
- Customer Credit Worthiness Check:** Three input fields labeled "Credit Rating", "Authorised By", and "Date", followed by a "Protect" button.
- Customer Risk Assessment:** Three input fields labeled "Risk Rating", "Authorised By", and "Date", followed by a "Protect" button.

The browser's status bar at the bottom shows "Done", "Local intranet", and "100%".

Figure 6.7: Screenshot of protected data items on form.

Before closing the template file, the user clicks the Save button (shown in Figure 6.6) to save any changes made to the document data. This action updates the XML files representing the document data and document metadata. The user must now update the workflow document ZIP file (i.e. replacing the old XML files with the new ones) before uploading it to KnowledgeTree. Similar to the check-out process discussed previously, this document check-in process is performed manually (although ideally, it would be completely automated in an actual implementation). The user ends the Shibboleth authenticated session by closing the web browser.

Once the workflow document is successfully checked in, the user invokes a workflow transition via the KnowledgeTree user interface. This moves the document to the next user in the workflow and also sends an email alert to inform the next user that the document is ready to check-out for processing.

6.1.5 Discussion

This case study shows how access control enforcement (applied to the data level) can be embedded within documents assigned to workflows. Although our implementation approach works, we encountered several issues whilst conducting the case study. This section discusses these issues and how our implementation could be improved.

The decision to use Shibboleth for identity management greatly simplified the user authentication aspect of the implementation, but needed considerable configuration during setup. Despite the initial high configuration effort and some minor factors affecting its maintenance (e.g. maintaining the LDAP service, time synchronisation, etc.), very little glue code (2 lines) was needed to extract the identity values supplied by Shibboleth for the Document Processing Environment. In all, our experiences with Shibboleth found it to be highly customisable and not tightly integrated with any of the prototype's other components. This made Shibboleth well suited as a pluggable identity management tool for this case study.

In contrast, minimal configuration was required to set up KnowledgeTree as our workflow engine. User management and workflow configuration were the only settings that needed attention. Our decision not to integrate KnowledgeTree with Shibboleth and the Document Processing Environment removed much of the complexity from the setup. Although this simplified the prototype, it meant that manually initiated connections (by the end user) to the KnowledgeTree server were required, hence

making the workflow execution more cumbersome and less automated.

Configuring KnowledgeTree as a web service would provide a better solution. This would allow the Document Processing Environment to automatically connect and retrieve the document from the workflow engine (after establishing an authenticated Shibboleth user session). Similarly, after the document has been processed, it could be automatically returned to the workflow engine, ready for the next user. Such modifications would remove the need for the user to directly interact with the workflow engine.

For the key server, we developed this ourselves in order to properly implement the shared key exchange algorithms proposed in Chapter 3. We used two encryption algorithms - RSA [56] (for asymmetric encryption) and AES [26] (for symmetric encryption). The Java Cryptography Architecture API provided an implementation of each.

Whilst testing the key server, we noticed that it returned a different ciphertext value each time an identical request was made. Interestingly, decrypting each of these ciphertext values resulted in the same plaintext value. This behaviour was caused by the RSA implementation, presumably to prevent replay attacks from occurring. The AES implementation, on the other hand, did not exhibit this same behaviour (i.e. a plaintext value always returned the same ciphertext value when encrypted with the same symmetric encryption key).

Implementing the shared keys using the expressions defined in the architecture model did not go without problem. The AES implementation insisted on 128 bit keys, but our key creation algorithm produced keys of varying lengths (depending on the key subject value). So that only 128 bits were produced, it was necessary to pad (or crop) the shared keys to ensure that the correct length was achieved.

Finally, creating the workflow document template and its associated scripts required the most programming effort - largely because we completely customised this component, making it tightly coupled to our document data design for this case study. Despite this, we did notice patterns emerge within the template, suitable for reuse in future case studies. Examples of such patterns are briefly mentioned in Section 6.1.4. By extracting these common patterns, this would simplify the development of templates for other scenarios, reducing the development time and programming effort. We intend to test this claim in our next case study.

6.2 Case Study 2 – Electronic Health Records

The inspiration for this next case study comes from the National E-Health Transition Authority (NEHTA) [49] and their research in formulating interoperable e-health services and architectures. The scenario is based on a patient preparing to undergo surgery, and focuses on the patient’s individual electronic health record (IEHR). The IEHR is edited and passed between multiple health practitioners throughout the workflow.

We expand on the previous case study in two respects. Firstly, we introduce concurrency into the workflow. The workflow for processing the loan application in the previous case study was sequential. For this case study, we model concurrent flows to demonstrate parallel execution. This requires us to add split and join operations to the workflow. The other feature that we introduce is constraints on role enactment. We define policies that specify the conditions that must be satisfied for a user to enact a particular role at a given workflow state. For example, such policies are applicable for defining *separation of duty* constraints.

The workflow starts with the patient visiting their general practitioner (GP) for an initial consultation. The patient is then referred to a pathologist and radiologist for tests before the operation. The test results are recorded in the patient’s IEHR and viewed by the surgeon performing the operation. After completing the operation, a surgeon’s report is appended to the IEHR and the patient is referred to a pathologist and radiologist for further tests. Again, the results from these tests are added to the IEHR. The final step is a follow-up consultation with the GP, where the doctor checks all of the test results and the surgeon’s report. An illustration of this workflow is given in Figure 6.8.

6.2.1 Process

We now discuss the workflow in greater detail and how it is supported using the prototype implementation.

1. The patient visits their GP for an initial consultation. During this visit, the GP adds the following to the patient’s IEHR: *referral* (to surgeon), *pathology request* and *radiology request*. The GP instructs the patient to visit a pathologist and radiologist for pre-operation and post-operation tests. The GP protects the referral so that it is read-only for surgeons and not accessible for

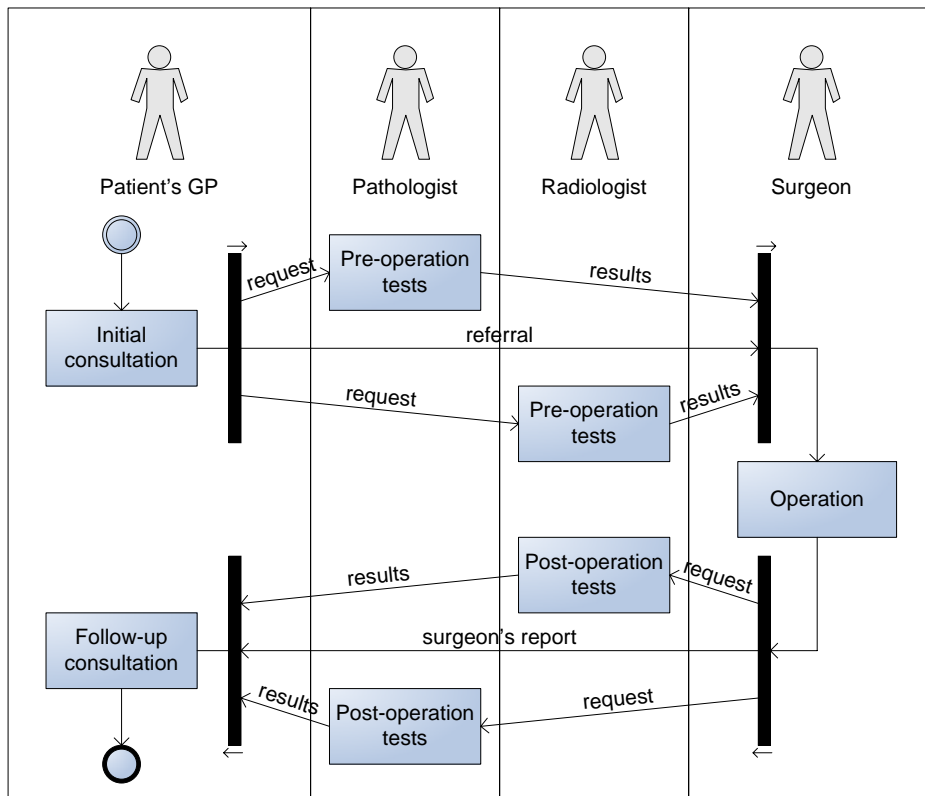


Figure 6.8: Medical record workflow.

others. The pathology and radiology requests are readable for all.

We assume the existence of an authority for storing and distributing EHRs to health practitioners on request. For the prototype, the document management system (DMS) serves as this authority.

The GP logs into the DMS and retrieves the patient's IEHR. In addition to adding the items mentioned above, the GP initiates the workflow for the IEHR to follow. This is achieved by the GP starting the workflow execution when the IEHR is returned to the DMS.

2. The next step involves concurrency. The patient visits a pathologist and radiologist for pre-operation tests. However, the order in which these tests are performed is irrelevant. This step is only completed after both tests have been performed and the results have been recorded to the IEHR. Both the pathologist and the radiologist record the results for their respective tests. The test results are protected so that only the patient's GP and surgeons can read them.

Similarly to the previous step, the pathologist/radiologist logs into the DMS and retrieves the patient's IEHR. The GP's request is viewed, the relevant test is performed and the results from the test are recorded to the IEHR before it is returned to the DMS.

3. Once both sets of test results are recorded, the surgeon accepts the IEHR and performs the operation. On the completion of this task, a surgeon's report is added to the IEHR. This report includes information on the procedure performed, surgeon's notes, etc. and is intended for the patient's GP. Hence it is protected as read-only for the patient's GP and not accessible for others.

But for the surgeon to receive the patient's IEHR, it needs to be retrieved from the DMS. The surgeon logs into the DMS and checks-out the IEHR. When finished with the record, the surgeon returns it to the DMS for the next workflow participant to retrieve.

4. After the operation, the patient undergoes further pathology and radiology tests (as requested by the patient's GP in step 1). These activities are concurrent as before. In fact, this step is almost identical to the second step in the workflow, but with two exceptions. Firstly, the patient need not visit the same pathologist or radiologist as before. Secondly, the results from these tests are protected slightly differently in that only the patient's GP can read them. Apart from these differences, retrieving and processing the IEHR remains the same as in the second step.

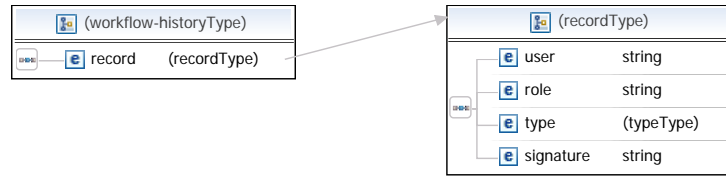


Figure 6.9: Workflow history section in document metadata component.

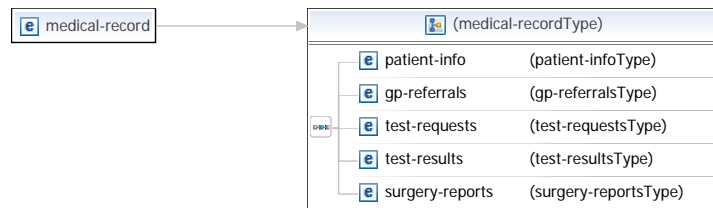


Figure 6.10: Top-level view of medical record schema.

5. Finally, the patient returns to the GP for a follow-up consultation. The results from the post-operation pathology and radiology tests must be available in the IEHR before the GP accepts the patient. An added constraint is applied whereby the GP who provided the initial consultation must be the same GP providing the follow-up consultation.

During the consultation, the GP retrieves the patient’s IEHR from the DMS and views both sets of test results and the surgeon’s report. When finished with the record, the GP returns it to the DMS and the workflow is concluded.

6.2.2 Document Design

The document data design for this case study is influenced by NEHTA’s National Discharge Summary design [50]. Similarly to the previous case study, we illustrate the IEHR design using XML schemas.

The document metadata design discussed in Section 6.1.2 remains largely unchanged except for the addition of a *workflow-history* section. The workflow history for this case study follows the design discussed in Section 3.10. It consists of a sequence of workflow history *records*, with each record containing the user who added the record, the user’s roles, the record type (either *acquire* or *release*) and the signature (created by the identity provider). The schema design for the workflow history metadata is illustrated in Figure 6.9.

The top-level view of the design is given in Figure 6.10. This diagram shows the IEHR composed of five sections:

1. *patient-info* contains information on the patient such as their name, patient ID, address, etc. This section can be read by any health practitioner who receives the IEHR.
2. *gp-referrals* contain referrals from the patient's GP instructing the patient to seek treatment from a more specialised health professional (e.g. surgeon, specialist, etc.). An individual referral contains the date, to whom the patient is referred to, the service requested and additional notes by the GP.
3. *test-requests* contain requests for the patient to undergo certain medical tests (e.g. blood tests, x-rays, etc.). An individual test request contains the date, the name of the test requested, the health practitioner requesting the test and any additional notes by this health practitioner.
4. *test-results* contain the results from medical tests that the patient has had. A test result report includes the date, the test performed, the health practitioner who performed the test and the actual test results.
5. *surgery-reports* are issued by surgeons on the completion of an operation. A surgery report includes the date, the name of the procedure performed, the name of the surgeon who performed the procedure and additional notes (e.g. required follow-up care, etc.).

6.2.3 Users and Roles

As discussed in Section 6.2, the roles involved in the workflow are: *Patient's GP*, *Pathologist*, *Radiologist* and *Surgeon*.

The role *Patient's GP* actually represents a single user who is a member of the role *General Practitioner*. As the name implies, this user is the patient's preferred general practitioner. The reason for creating a special role for this user is because shared keys are assigned to roles (not users). Hence it was necessary to model the case study this way.

The patient's GP processes the IEHR at two separate stages in the workflow (as shown in Figure 6.8). A new feature introduced to this case study is *constraints on role enactment*. We demonstrate

this by applying policies to the workflow activities assigned to *Patient's GP*. If the user attempting to perform the follow-up consultation was not the same user who performed the initial consultation, then the user is not permitted to perform the follow-up consultation. In contrast, we do not apply constraints on the roles *Pathologist* and *Radiologist*. For instance, the user performing the pre-operation test need not be the same user performing the post-operation test.

The user to role mappings defined for this case study are as follows:

- user *alfa* acts as the patient's GP
- user *bravo* acts as the patient's GP (secondary user to test role constraints)
- user *charlie* acts as the pathologist
- user *delta* acts as the radiologist
- user *echo* acts as the surgeon

Similarly to the first case study, each of these users has their own public/private keypair plus a login account with the Shibboleth identity provider and the KnowledgeTree DMS.

6.2.4 Implementation

This case study extends the prototype implementation used in the previous case study, but retains compliance with the architecture model and workflow object model specified in Chapters 3 and 4 respectively.

To recap the implementation setup described in the first case study, it consists of three components: the Shibboleth identity provider, the KnowledgeTree DMS, and the document processing environment (DPE). The user authenticates himself to Shibboleth, retrieves the workflow document from KnowledgeTree, and views/edits it via the document template (running in the DPE). Shared keys associated with the workflow document are automatically retrieved by the DPE from the key servlet (located at the identity provider). When the user is finished with the workflow document, it is returned to KnowledgeTree and the next transition is invoked (sending the document to the next user in the workflow).

The implementation discussed in Section 6.1.4 is extended as follows:

1. The inclusion of *workflow history* in the document metadata component.
2. Support for *document checking* (performed by verifying workflow participants).
3. Workflows containing *concurrent workflow activities*.

To implement support for workflow history, it was necessary to extend the identity provider's functionality to include a new servlet for accepting requests to update the document's workflow history. The client-side scripts for the document template were also extended to allow for the workflow history to be automatically updated on load and save events. That is, when the user loads the workflow document after retrieving it from KnowledgeTree, a request is automatically sent to the identity provider for an *acquire* history signature. When the user saves and releases the workflow document, a second request is automatically sent to the identity provider for a *release* history signature. Both signatures are appended to the existing workflow history in the workflow document's metadata.

For document verification, we conducted a simple test that checked that the role constraint (i.e. the user who performed the follow-up consultation must be the same user who performed the initial consultation) was upheld. Given that both of these workflow activities are assigned to the role *Patient's GP* (which is defined as a "single user" role), this should ensure that this constraint is always satisfied. But for testing purposes, we assigned a second user to this role to demonstrate how a violation of the role constraint is detectable.

Although a verifying workflow participant is not illustrated in Figure 6.8, we consider their engagement to occur at the conclusion of the workflow. Normally this participant would be modelled into the workflow description but has been excluded in this case for brevity.

To achieve this, we applied an XML stylesheet to the document metadata. The XML stylesheet defined XPath expressions for extracting the workflow history concerning the initial and follow-up consultations. If the users listed in these extracted workflow history records are equal, the stylesheet verifies that the constraint is upheld. Otherwise, the stylesheet displays the conflicting user names. Before performing this check, the verifying workflow participant should verify the workflow history signatures to ensure that the information given in the workflow history is reliable. If checking multiple constraints, a separate XML stylesheet needs to be defined for each test case.

Figure 6.11 shows an example user interface displaying the test applied and the result (for both

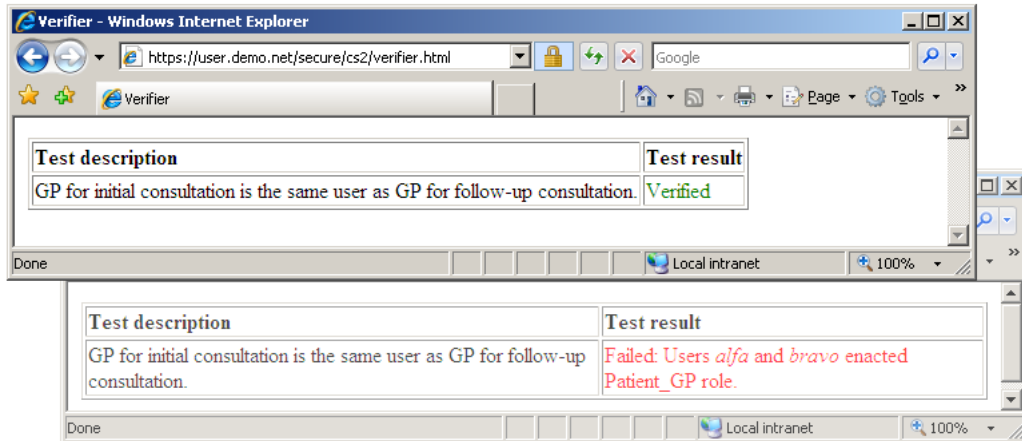


Figure 6.11: Screenshots of verifier interface.

a successful and failed test). This simple HTML page uses client-side JavaScript to load the XML stylesheet and document metadata as DOM objects, before applying the XML stylesheet to the metadata and displaying the outcome (i.e. the test result) on the HTML page.

For concurrent workflow activities, parallel execution is tool dependent. In the case of KnowledgeTree, it is not supported. To overcome this, we remodelled the workflow description to simulate parallel execution by using selection operators (with each possible execution combination of the concurrent workflow activities existing as a separate flow path). Despite the workflow activities not actually executing simultaneously, the same outcome is produced after all concurrent activities have been accomplished. Such a tactic is possible in this example since each concurrent activity is concerned with separate sections of the workflow document. Regardless of the order in which these concurrent activities are executed, they have no impact on one another's execution. Hence this technique is adequate for independent concurrent workflow activities.

The remodelled workflow is illustrated in Figure 6.12. Decision points are shown where the workflow previously had split operators. Join operators are not required as the remodelled workflow does not contain concurrently executing flow paths (and hence multiple copies of the same workflow document instance) to synchronise.

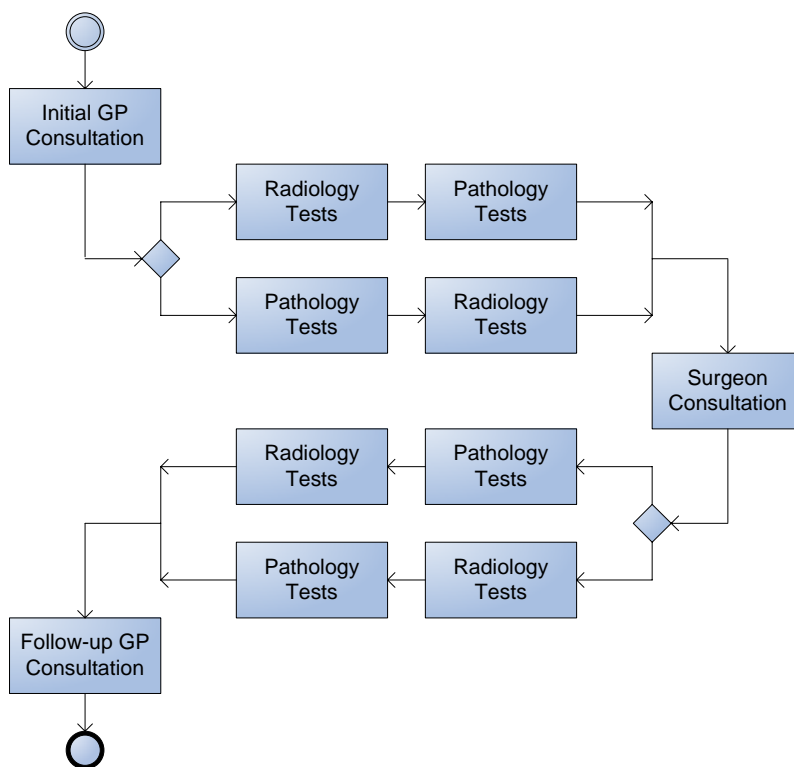


Figure 6.12: Remodelled medical record workflow.

6.2.5 Discussion

We emphasise that the overall aim of these case studies is to demonstrate the capabilities of the architecture model and workflow object model, and not to show the most optimal implementation for these models. The medical record case study continues to fulfil this objective by expanding on the first case study – most notably, in its support for concurrent workflow activities and constraints on role enactment. Furthermore, this case study identifies those parts from the previous case study that were easily transferable to help lower the implementation effort. This section discusses both of these aspects, focusing on the issues and experiences encountered whilst conducting the medical record case study.

It is important to reiterate that although this case study expands on the first one, it continues to comply with the architecture model and workflow object model specifications, despite having to change the prototype implementation to support the expansions. These changes mainly concerned introducing workflow history support. Not only did this require a new servlet for the identity provider (to accept and process workflow history requests), but also modifications to the scripts in the document processing environment (so that workflow history records could be automatically appended to the document metadata). The effort required to introduce this functionality to the prototype was actually quite minimal. Given that we already had the *CreateMetadataEntry* servlet (from the document processing environment), we were able to adapt much of this code to the workflow history servlet. Similarly, we could easily adapt existing scripts to automatically request workflow history records and append them to the document metadata.

Section 6.2.4 discussed how the workflow description was remodelled to simulate parallel execution, since KnowledgeTree does not support concurrent workflow activities. This removed the need to synchronise changes to multiple copies of the workflow document as the remodelled workflow description did not contain concurrently executing flow paths. In the case of concurrent workflow activities, it is important that disjoint sections of the workflow document are modified to avoid synchronisation conflicts. Note also that alternative tools may exist which provide a more extensive range of workflow modelling constructs than KnowledgeTree. Although we were not constrained to using KnowledgeTree, we retained its use for the medical record case study for simplicity.

Introducing a verifying workflow participant to this case study was needed for checking the

role enactment constraints imposed on the workflow document. As discussed previously, an XML stylesheet was used to define the test case to perform this check. Given that the check performed was relatively simple, XML stylesheets (including XPath expressions) suffice in this instance. But for more complex tests, this method may not be suitable. For example, should the test case require the use of variables, a scripting language such as JavaScript would be better suited as it offers more programming language constructs and functions than XML stylesheets. Alternatively, a separate tool could be plugged into the prototype to verify the workflow document.

A point worth noting is that all data to be checked must be contained in the workflow document (i.e. in the document data or document metadata). Another point is that checking occurs *after the fact*. In other words, the (potentially prohibited) event occurs first and then it is checked later. Because of the architecture model's decentralised nature, there is no single authority to control and prevent prohibited events. But it is critical that should a prohibited event occur, it must be detectable by the verifying workflow participant. If a prohibited event is detected, any attempt to restore the workflow document to a valid state depends on the workflow description (i.e. whether recovery flow paths are specified in the workflow). However, it must be noted that since checking occurs *after the fact*, this model is not suitable for potentially life-endangering workflows (e.g. when prohibited events could lead to fatal outcomes).

In contrast to the previous case study, creating the workflow document template (including its associated scripts) required little programming effort given that much of the code from the first case study could be reused with minimal alteration. This confirms the claim made in Section 6.1.5 that the development time and programming effort would be reduced by extracting common patterns in the templates.

Patterns that were reused in this case study included the methods *loadWorkflowDocument*, *protect-DataItem* and *saveWorkflowDocument* (although minor changes relating to the document data structure and template were required). The Java Servlets did not require any modifications except for defining the private keys (as new user identities were used in this case study). For demonstration purposes, private keys were statically defined in the servlets. An actual deployment would require servlets to retrieve private keys from a separate source (as opposed to being embedded within the servlets).

Other code, such as initialising the template with the data values from the document data, required adjustments to reflect the changed document data schema (i.e. the medical record schema defined in Section 6.2.2). But in all, this case study has clearly demonstrated that the workflow document template code can be largely reused in other workflow scenarios given the patterns that exist.

6.3 Case Study 3 – Wildlife Report

Already we have seen the architecture model demonstrated in two different scenarios. The first case study provided a *proof-of-concept* and suggested that patterns may exist in the architecture model's implementation and deployment. The second case study tested this claim and identified the parts that were reusable. In this third and final case study, we aim to validate that these parts are indeed reusable and to extract common patterns from all three case studies.

This case study is based on a scenario taken from the Wildlife Enforcement Monitoring System (WEMS) project at the United Nations University [84]. It looks at an e-government example, but is applicable for information sharing scenarios in general. The workflow document represents a wildlife report that is processed by and circulated amongst multiple government officials of varying levels within the government hierarchy. We specify access control policies (defining the access permissions for each role) and apply these to the report.

The workflow begins with a Field officer creating a new report and recording the personal data, location, species and any notes. The report is then transferred to a State office, and later to a National office, for processing. A verdict on the infringement is recorded by a Judge and the report is made available to International officers and Researchers.

Although a workflow exists, we concentrate entirely on the access control aspects of the case study. Workflow-related features are excluded from the discussion for brevity. This case study expands on the previous ones by introducing the ability to apply and enforce multiple access control permissions on a single data item.

6.3.1 Policies

We assign access control permissions to the following roles: *Field officer (FO)*, *State officer (SO)*, *National officer (NO)*, *International officer*, *Researcher* and *Judge*.

The access control policies are as follows:

- Field officers create a new report by recording the personal data, location, species and FO-notes. Once created, these data items must be *read only* for all roles (except for Researcher, who can only read the location and species).
- State officers can write SO-notes but can read all other data items.
- National officers can write NO-notes but can read all other data items.
- International officers have *read only* access for all data items.
- Only a Judge can write the verdict. State officers, National officers and International officers can only read it. A Judge can read all other data items.

In order for Field officers to write the personal data, location, species and FO-notes data items, we leave these items unprotected and introduce a new role, *Field officer verifier*, to protect these items after the Field officer has processed the report. This is required as the protector of a data item must have both read and write access to that data item.

6.3.2 Document Design

The document data consists of seven data items: *personal data*, *location*, *species*, *FO-notes*, *SO-notes*, *NO-notes* and *verdict*. The data values for verdict are limited to *alleged* (default value), *guilty* or *not-guilty*. Each of these data items are displayed on the template as text areas.

The document metadata largely follows the design used in the previous case studies. We modify the design for protected data item records to accommodate multiple access control permissions. A distinction is made between subjects with read access and subjects with write access. This is necessary as multiple access control permissions involve multiple shared keys for the one data item. Figure 6.13 illustrates the modified design for protected data item records.

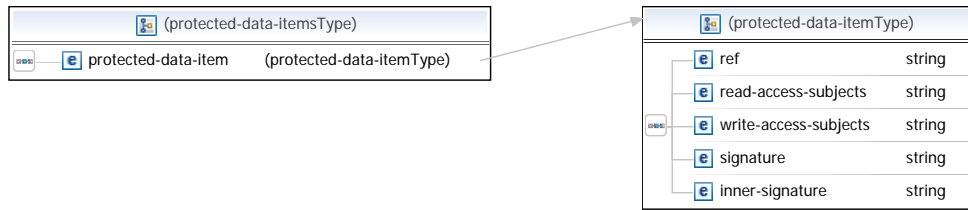


Figure 6.13: Modified design for protected data item records.

6.3.3 Implementation

Similarly to the previous case study, we again extend the prototype implementation, but still retain compliance with the architecture model and workflow object model specified in Chapters 3 and 4 respectively. This extension consists of the ability to apply and enforce multiple access control permissions on a single data item.

Implementing this extension required changes to the client-side scripts and the *CreateMetadataEntry* servlet to reflect the modified design for protected data item records and to support our encryption technique for multiple permissions (Section 3.10). Furthermore, the algorithm for the method that protects data items was slightly changed as two shared keys were now required for each protected data item. No changes were required for requesting or creating shared keys, as well as any encryption or decryption operations.

6.3.4 Discussion

This case study validated the claim confirmed by the second case study (and proposed after the first case study) that reusable patterns exist in the implementation of the architecture model and workflow object model. Additionally, the case study successfully demonstrated multiple permissions on individual data items using the extended prototype implementation. In this section, we compare the multiple permissions implementation with the access control implementation deployed in the previous case studies. We also discuss the common patterns that were validated as easily transferable and consequently, helped to lower the implementation effort.

Several advantages were evident using the multiple permissions implementation. With the access control implementation used in the previous case studies, it was necessary to determine which access

type was applied to each data item when loading and saving the workflow document (as a different encryption technique is used for each access type). Also, the metadata entries for *read write* protected data items are structured differently to the metadata entries for *read only* protected data items. This is not the case for the multiple permissions implementation, making processing the metadata entries less complicated. In fact, it is possible to use the multiple permission implementation to entirely implement the access control policies in the previous case studies.

A disadvantage of the multiple permissions implementation is that multiple shared keys are required to protect a single data item. This resulted in increased requests to the key server. However such requests could be optimised by requesting multiple keys in the one request to the key server. Despite there existing multiple keys, the multiple permissions implementation only uses symmetric encryption, compared to the *read only* encryption technique which uses a combination of symmetric and asymmetric encryption. As a result, the implementation used for this case study is faster.

6.4 Lessons Learned

We observed that the following components varied between the three case studies:

1. document data structure
2. user/role identities
3. workflow description
4. template form design
5. policy rules

The main reason for this is because these components are all application specific. In fact, policy rules can also vary between instances of the same scenario. The formation of these components is highly influenced by the scenario being modelled. But despite this, their implementation is not constrained by the scenario.

Components with similar formations across the case studies included:

1. document metadata structure

2. key creation and distribution
3. workflow history creation and distribution
4. template scripts (including backend servlets)

Although the document metadata structure changed slightly between case studies, these changes related to extensions to the prototype implementation and not because of changing scenarios. Overall, the document metadata structure remained largely the same. In contrast, key creation and distribution was consistent for each access type implemented. Workflow history records were introduced in the second case study and retained for the third case study without modification. These three components are generic by design, hence enabling them to be easily reused for different cases.

The template scripts and associated backend servlets also contained similarities across the three case studies but they did require modifications to reflect changes to the document data structure and template form design. Regardless of these changes, patterns emerged in the operations performed by these scripts and servlets.

The main patterns emerging from the scripts included:

1. *protect data item* – accepts the key subjects (including the respective access types) and data item to protect, then requests the appropriate shared key(s) before protecting the data item and creating a metadata entry for it.
2. *load workflow document* – unlocks each protected data item accessible to user before displaying unprotected data items and accessible protected data items on template form.
3. *save workflow document* – saves changes to unprotected data items and permissible changes to protected data items.

Additional patterns (existing within the main patterns) included *create metadata entry*, *append workflow history record*, *get shared key*, *encrypt data item* and *decrypt data item*. Unlike the main patterns, these patterns are independent from the document data structure and template form design. Hence they can be easily transferred and reused across different scenarios.

Comparing the template scripts from all three case studies, the only significant differences observed were the scripts for linking data items to form elements (and vice versa). This raises the

<i>Application specific components</i>	<i>Generic components</i>
document data structure	document metadata structure
user/role identities	key creation/distribution
workflow description	workflow history creation/distribution
template form design	template scripts
template linking scripts	
policy rules	

Table 6.1: List of application specific and generic components.

question of whether including such “linking” information within the workflow document would make the template scripts completely transferable and reusable without modification. That is, in reference to Figure 5.7 from Chapter 5, storing the linking information in the *document data* XML file instead of the *scripts.js* file. Such a move though would tightly couple the workflow document to a single template. By design, the workflow document is not intended to be restricted to the one template but compatible for multiple templates (presentation formats). Hence including this information may have an adverse effect and therefore needs further investigation.

To conclude, these case studies have identified those components of the architecture model and workflow object model that are reusable and those that are application specific. The patterns extracted from these case studies serve as general principles for the development of a new system implementing the models. Table 6.1 summarises the components identified as application specific or generic.

6.5 Summary

This chapter has reported on the three case studies performed to determine the practical feasibility of the proposed models. Not only have these case studies demonstrated potential applications for e-business, e-health and e-government, it has also identified patterns existing in the implementation of these models. As a result, further developments to the prototype towards a fully fledged product will be greatly assisted by our experiences described in this chapter. In the following chapter, we conclude the thesis by drawing several conclusions from this research and also describe possible future directions

that expand on this work.

Chapter 7

Conclusion

This thesis proposes two models – an architecture model and an workflow object model – to support cryptographic access control for inter-organisational collaborative environments. The models focus on securing sensitive data that (1) follows a pre-defined workflow and (2) involves multiple contributors (spread across different security domains). Our approach embeds access control enforcement within the workflow object rather than relying on individual security realms to enforce the access control policies. Access to sensitive data requires possession of the encryption key(s) linked to that data item. These keys are issued to authorised users on-demand by identity providers (who are also responsible for authenticating users). Verifiers, on the other hand, are tasked with checking the workflow object against security policies to ensure that contributors have not invalidated the workflow object's content.

Support for finer granularity access control is another feature of the proposed models. The intention behind this is to allow for tighter policy constraints on the sensitive data. The workflow object model views content as a structured collection of data items as opposed to a single data unit. This enables us to separately protect individual sensitive data items and to partition the workflow object (hence restricting different parts of the object to only certain contributors).

A prototype of these models was implemented using existing tools where possible. From this prototype, three case studies were performed to demonstrate the practical feasibility of the proposal. The case studies showed the models' applicability to areas in e-business, e-health and e-government. While the first case study was simply a proof-of-concept, the subsequent case studies were designed to identify reusable patterns that could be extracted and applied to different case study scenarios. As

well as confirming that access control enforcement can be embedded within documents assigned to workflows spanning across organisational boundaries, these case studies successfully demonstrated the applicability of this work to a broad range of application fields.

The architecture model and the workflow object model were designed to solve the research question of this thesis: How to restrict access to sensitive data following a workflow that spreads across organisational boundaries, in a fashion that is independent from the underlying infrastructure of each organisation, not restricted to any particular implementation software and does not rely on each organisation's security realm to enforce the access control policies? These requirements have been addressed and accomplished in the design of the models.

7.1 Contributions

The contributions directly respond to the research question of this thesis. This work targets the practical side of securing information flow for inter-organisational collaborative environments. Where much of the existing work focuses on analysis and the theoretical aspects in this field, this thesis contributes in terms of implementation and practical feasibility in various application domains. This contribution is realised in the following deliverables of this thesis:

- An architecture model to support secure information flow for inter-organisational collaborative environments has been designed. We do not align this model to any specific application domain. It is generic by design so that it can be readily applied to a number of situations.

The architecture model uses Role-Based Access Control (RBAC) [59] for specifying access control restrictions, but relies on encryption algorithms to enforce these restrictions. Key exchange protocols are defined for requesting shared keys from identity providers and releasing these keys to authorised users. The model also allows the workflow object's history to be recorded so that the produced audit trail can be used for verification purposes (e.g. checking that the workflow object conforms to verifiers' policies). Several basic access permission types are currently supported, paving the way for more complex permissions to be developed. Furthermore, the model does not insist on a specific transfer medium, therefore allowing various transfer types to be used (e.g. email, Web services, document management systems, etc.).

- A design for a workflow object model that encapsulates data assigned to a workflow within an inter-organisational environment. The workflow object is comprised of several components, with the object data and object metadata forming the core components. We found that representing these core components in XML provides an ideal format for realising their formation. This is largely because of the hierarchical data structure (evident in both components) which was influenced by the W3C's Document Object Model [18]. Other contributing factors for using XML include (1) it is a machine interpretable format (hence automated processing can be facilitated) and (2) it can easily represent data items as defined in Chapter 4.
- A prototype implementation of the architecture model and workflow object model has been constructed. Although this prototype does not fully implement both of these models, it provides an extensible platform for experimenting with the access control capabilities and features of the models. So that this prototype can be reused to test different scenarios, the implementation design is flexible, "plugging in" existing tools where possible. Achieving true "plug and play" is not possible since glue code will inevitably be required to connect the components together.
- Case studies have been performed to demonstrate the practical feasibility and applicability of these models to a range of application domains. The experiences encountered from these case studies revealed reusable patterns existing across all three scenarios modelled. After identifying and validating this claim, the case studies allowed us to categorise application specific components and generic components.

7.2 Future Work

The models proposed in this thesis achieve their objective of facilitating secure information flow in inter-organisational collaborative environments. These models not only expand on the efforts of existing work in cryptographic access control, but they also provide a foundation for further work. Several directions for extending and improving the models and implementation are suggested:

- The models define how data is protected within the object data component of the workflow object. As for securing the metadata, no protection strategies are defined. This is a major

limitation since it allows attackers the ability to change metadata values without detection. For example, changing a key subject value could result in the incorrect shared key being retrieved, resulting with the user being denied access to the data item. Therefore, in addition to applying this extension, a thorough, formal analysis is required.

- After access control policies have been created and applied to the workflow object, the ability to modify them is currently not supported. Hence there is a need to extend the models to support this. Also required are *meta policies* that specify the users who are authorised to create, modify and delete a policy applied to a particular protected data item. This ability would give data owners tighter control over their data items within the workflow object. However, enforcing such policies beyond organisational boundaries would be a non-trivial and challenging issue. It is unclear how cryptographic protection alone would suffice to achieve this.

Expanding on this, complex access control permissions could be introduced. For example, a set of access control permissions become enforced only after certain conditions are satisfied later in the workflow. Such conditional access control capability provides for tightened security policies to be applied to data items.

- Adapting the implementation design to operate over service-oriented and event-driven architectures is another future direction. Given that these architectures are synonymous with distributed environments and collaboration, there is the potential that this research could have widespread applications in services delivery platforms. For instance, investigating the feasibility of the proposed models for Web 2.0 and social networking applications could reveal new areas for further deployment.

As a result of these future directions, a more comprehensive solution would be realised for accomplishing secure information flow for inter-organisational collaborative environments.

Appendix A

Acronyms

ACL	Access Control List
AES	Advanced Encryption Standard
API	Application Programming Interface
BLP	Bell-LaPadula model
BPEL	Business Process Execution Language
CA	Certification Authority
CAS	Central Authentication Service
COM	Component Object Model
CORBA	Common Object Requesting Broker Architecture
DAC	Discretionary Access Control
DES	Data Encryption Standard
DMS	Document Management System
DOM	Document Object Model
DPE	Document Processing Environment
DRM	Digital Rights Management
EBNF	Extended Backus-Naur Form
EJB	Enterprise JavaBeans

APPENDIX A. ACRONYMS

HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IEHR	Individual Electronic Health Record
KDC	Key Distribution Centre
LDAP	Lightweight Directory Access Protocol
MAC	Mandatory Access Control
MD5	Message-Digest algorithm 5
MLS	Multi Level Security
NEHTA	National E-Health Transition Authority
NIS	Network Information Service
OASIS	Organization for the Advancement of Structured Information Standards
ORCON	Originator Controlled
PAP	Policy Access Point
PDF	Portable Document Format
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PMA	Policy Management Authority
RBAC	Role-Based Access Control
RSA	Rivest-Shamir-Adleman
SAML	Security Assertion Markup Language
SHA	Secure Hash Algorithm
SOAP	Simple Object Access Protocol
SoD	Separation of Duty
SQL	Structured Query Language
SSO	Single Sign On

APPENDIX A. ACRONYMS

UCON	Usage Control
UDDI	Universal Description, Discovery and Integration Protocol
W3C	World Wide Web Consortium
WfMC	Workflow Management Coalition
WfMS	Workflow Management System
WS	Web Services
WSDL	Web Services Description Language
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language

Appendix B

Source Code

The prototype implementation described in Chapter 5 is designed for experimentation purposes. For this reason, inclusion of the source code in this thesis would be inappropriate considering the unrefined and experimental nature of the prototype.

If you would like to experiment with this system, please contact the author for a copy of the source code. The source code is available as “open source”, licensed under the GNU General Public License Version 3[†].

[†]The GNU General Public License Version 3 is available at <http://www.gnu.org/licenses/gpl-3.0.txt>

Appendix C

Publications

C.1 Publications by the Candidate Relevant to the Thesis

1. S. Bracher and P. Krishnan. (2008). Implementing Secure Document Circulation: A Prototype. In *SAC '08: Proceedings of the 23rd Annual ACM Symposium on Applied Computing*, pages 1452–1456, Fortaleza, Brazil, March 2008. ACM ISBN 1-59593-753-7
2. S. Bracher and P. Krishnan. (2008). Secure Document Circulation: An Architecture for e-Health (revised ehPASS paper). In *electronic Journal of Health Informatics* (Special Issue on Privacy and Security), 3(1): e2, pages 1–9. ISSN 1446-4381 <http://ejhi.net/>
3. S. Bracher and P. Krishnan. (2006). Secure Document Circulation: An Architecture for e-Health. In *ehPASS '06: Proceedings of the National e-Health Privacy and Security Symposium*, pages 67–77, Brisbane, Australia, October 2006. QUT ISBN 174107 138 0
4. S. Bracher. (2005). Secure Document Circulation Using Web Services Technologies. In *ICSOC '05: Proceedings of the IBM PhD Student Symposium at the 3rd International Conference on Service Oriented Computing*, pages 31–36, Amsterdam, The Netherlands, December 2005. CEUR Workshop Proceedings Vol-169 ISSN 1613-0073
<http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-169/>

C.2 Additional Publications by the Candidate Relevant to the Thesis but not Forming Part of it

1. A. Khurat, J. Abendroth, S. Bracher and P. Krishnan. (2007). Towards Client Privacy Policy Enforcement for Small-Medium Enterprises. In *Proceedings of the Inaugural IEEE International Conference on Digital Ecosystems and Technologies*, pages 319–324, Cairns, Australia, February 2007. IEEE ISBN 1-4244-0470-3
2. S. Bracher and P. Krishnan. (2005). Enabling Security Testing from Specification to Code. In *IFM '05: Proceedings of the 5th International Conference on Integrated Formal Methods*, pages 150–166, Eindhoven, The Netherlands, November/December 2005. Springer-Verlag, LNCS 3771.
3. D. Huang and S. Bracher. (2005). Towards Evidence-based Trust Brokering. In *SECOVAL '05: Proceedings of the 1st IEEE SECOVAL Workshop on the Value of Security through Collaboration*, pages 43–50, Athens, Greece, September 2005. IEEE ISBN 0-7803-9469-0

Bibliography

- [1] S. Akl and P. Taylor. Cryptographic Solution to a Problem of Access Control in a Hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, 1983.
- [2] J. P. Anderson. Computer Security Planning Study. Technical Report 73-51, Air Force Electronic System Division, 1972.
- [3] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Unified Exposition and MULTICS Interpretation. Technical Report 75-306, The MITRE Corporation, 1976.
- [4] E. Bertino, S. Castano, and E. Ferrari. Securing XML Documents with Author-X. *IEEE Internet Computing*, 5(3):21–31, 2001.
- [5] K. J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report 76-372, The MITRE Corporation, 1977.
- [6] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2003.
- [7] D. Brewer and M. Nash. The Chinese Wall Security Policy. In *IEEE Symposium on Security and Privacy*, pages 206–214, 1989.
- [8] S. Campadello, L. Compagna, D. Gidoin, S. Holtmanns, V. Meduri, J.-C. Pazzaglia, M. Seguran, and R. Thomas. A7.D1.1 – Scenario selection and definition. Technical report, SERENITY (System Engineering for Security and Dependability), 2006. http://serenity-forum.org/Work-package-7-1.html?artsuite=0#sommaire_1.

BIBLIOGRAPHY

- [9] J. Cardoso, R. Bostrom, and A. Sheth. Workflow Management Systems and ERP Systems: Differences, Commonalities, and Applications. *Inf. Technol. and Management*, 5(3-4):319–338, 2004.
- [10] Central Authentication Service. <http://www.ja-sig.org/products/cas/index.html>. JA-SIG, 2006.
- [11] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *IEEE Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, 1987.
- [12] W. R. Cook, S. Patwardhan, and J. Misra. Workflow Patterns in Orc. In *COORDINATION '06: Proceedings of the 8th International Conference on Coordination Models and Languages*, volume 4038 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 2006.
- [13] J. Crampton, K. Martin, and P. Wild. On Key Assignment for Hierarchical Access Control. In *CSFW '06: Proceedings of the 19th IEEE workshop on Computer Security Foundations*, pages 98–111, Washington, DC, USA, 2006. IEEE Computer Society.
- [14] B. Curtis, M. I. Kellner, and J. Over. Process Modeling. *Commun. ACM*, 35(9):75–90, 1992.
- [15] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Design and Implementation of an Access Control Processor for XML Documents. In *Proceedings of the 9th International World Wide Web Conference on Computer Networks : The International Journal of Computer and Telecommunications Networking*, pages 59–75, Amsterdam, The Netherlands, 2000. North-Holland Publishing Co.
- [16] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. A Fine-Grained Access Control System for XML Documents. *ACM Trans. Inf. Syst. Secur.*, 5(2):169–202, 2002.
- [17] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. A Data Outsourcing Architecture Combining Cryptography and Access Control. In *CSAW '07: Proceedings of the 2007 ACM workshop on Computer Security Architecture*, pages 63–69, New York, NY, USA, 2007. ACM Press.

BIBLIOGRAPHY

- [18] Document Object Model. <http://www.w3.org/DOM/>. W3C, 2005.
- [19] Extended Backus-Naur Form – International Standard ISO/IEC 14977: 1996(E). [http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip). ISO/IEC, 1996.
- [20] eXtensible Access Control Markup Language (XACML) (version 2.0). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml. OASIS, 2005.
- [21] S. Farrell and P. Kaijser. A Non-repudiation Service Architecture and Certification Infrastructure. In *EDITT Workshop*, pages 113–124, 1995.
- [22] D. Ferraiolo, J. Cugini, and R. Kuhn. Role-Based Access Control (RBAC): Features and Motivations. In *Annual Computer Security Application Conference*, 1995.
- [23] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.
- [24] M. Fetscherin and M. Schmid. Comparing the Usage of Digital Rights Management Systems in the Music, File and Print Industry. In *Proceedings of the 2003 International Conference of Electronic Commerce*, pages 316–325. ACM Press, 2003.
- [25] FIPS PUB 180-2 – Specification for the Secure Hash Signature Standard (SHS). <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf>, 2002. National Institute of Standards and Technology (NIST).
- [26] FIPS PUB 197 – Specification for the Advanced Encryption Standard (AES). <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>. National Institute of Standards and Technology (NIST), 2001.
- [27] W. Fumy and J. Sauerbrey. *Enterprise Security: IT Security Solutions: Concepts, Practical Experiences, Technologies*. John Wiley & Sons, 2006.
- [28] D. Georgakopoulos. Teamware: An Evaluation of Key Technologies and Open Problems. *Distrib. Parallel Databases*, 15(1):9–44, 2004.

BIBLIOGRAPHY

- [29] V. Gligor. Characteristics of Role-Based Access Control. In *RBAC '95: Proceedings of the 1st ACM Workshop on Role-Based Access Control*, page 10, New York, NY, USA, 1996. ACM Press.
- [30] GNU Privacy Guard. <http://www.gnupg.org/>. Free Software Foundation, 2004.
- [31] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to Web Services Architecture. *IBM Systems Journal*, 41(2):170–177, 2002.
- [32] R. Graubart. On the Need for a Third Form of Access Control. In *Proceedings of the 12th National Computing Security Conference*, pages 296–303, 1989.
- [33] M. Hafner, R. Breu, and M. Breu. A Security Architecture for Inter-Organizational Workflows: Putting Security Standards for Web Services together. In *ICEIS 2005: Proceedings of the 7th International Conference on Enterprise Information Systems*, pages 128–135, 2005.
- [34] A. Harrington and C. Jensen. Cryptographic Access Control in a Distributed File System. In *SACMAT '03: Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, pages 158–165, New York, NY, USA, 2003. ACM Press.
- [35] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996.
- [36] Java Cryptography Architecture Reference Guide. <http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html>. Sun Microsystems, 2006.
- [37] JBoss jBPM Workflow Management System. <http://www.jboss.com/products/jbpm>. Red Hat Middleware, 2007.
- [38] M. Kang, J. Park, and J. Froscher. Access Control Mechanisms for Inter-Organizational Workflow. In *SACMAT '01: Proceedings of the 6th ACM Symposium on Access Control Models and Technologies*, pages 66–74, New York, NY, USA, 2001. ACM Press.
- [39] KnowledgeTree Document Management System (Open Source Edition) (version 3.4.2). <http://www.knowledgetree.com/>. JamWarehouse, 2007.

BIBLIOGRAPHY

- [40] H. Koshutanski and F. Massacci. An Access Control Framework for Business Processes for Web Services. In *XMLSEC '03: Proceedings of the 2003 ACM Workshop on XML Security*, pages 15–24, New York, NY, USA, 2003. ACM Press.
- [41] R. Kraft. Designing a Distributed Access Control Processor for Network Services on the Web. In *XMLSEC '02: Proceedings of the 2002 ACM Workshop on XML Security*, pages 36–52, New York, NY, USA, 2002. ACM Press.
- [42] S. H. Kwok. Digital Rights Management for the Online Music Business. *SIGecom Exch.*, 3(3):17–24, 2002.
- [43] The Liberty Alliance Project. <http://www.projectliberty.org/>. 2001–2008.
- [44] Q. Liu, R. Safavi-Naini, and N. P. Sheppard. Digital Rights Management for Content Distribution. In *ACSW Frontiers '03: Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003*, pages 49–58, Darlinghurst, Australia, 2003. Australian Computer Society, Inc.
- [45] T. S. Messerges and E. A. Dabbish. Digital Rights Management in a 3G Mobile Phone and Beyond. In *DRM '03: Proceedings of the 3rd ACM Workshop on Digital Rights Management*, pages 27–38, New York, NY, USA, 2003. ACM Press.
- [46] G. Miklau and D. Suciu. Controlling Access to Published Data Using Cryptography. In *VLDB '03: Proceedings of the 29th International Conference on Very Large Data Bases*, pages 898–909. VLDB Endowment, 2003.
- [47] J. Misra and W. Cook. Computation Orchestration: A Basis for Wide-area Computing. *Software and Systems Modeling (SoSyM)*, 6(1):83–110, March 2007.
- [48] MySQL Database Server. <http://www.mysql.com/>. MySQL AB, 2007.
- [49] National E-Health Transition Authority (NEHTA). <http://www.nehta.gov.au/>. 2007.
- [50] NEHTA Industry Seminar: Jurisdiction Summit and Vendor Focus Sessions (Discharge Summary: The Information – Sample Discharge Summary). http://www.nehta.gov.au/index.php?option=com_docman&task=doc_details&gid=309&Itemid=139&catid=150. 2007.

BIBLIOGRAPHY

- [51] Organization for the Advancement of Structured Information Standards (OASIS). <http://www.oasis-open.org/>. 1993–2008.
- [52] J. Park and R. Sandhu. Originator Control in Usage Control. In *POLICY '02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks*, pages 60–66, Washington, DC, USA, 2002. IEEE Computer Society.
- [53] J. Park and R. Sandhu. Towards Usage Control Models: Beyond Traditional Access Control. In *SACMAT '02: Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, pages 57–64, New York, NY, USA, 2002. ACM Press.
- [54] D. Riehle and H. Züllighoven. Understanding and Using Patterns in Software Development. *Theory and Practice of Object Systems*, 2(1):3–13, 1996.
- [55] R. Rivest. RFC 1321 – The MD5 Message-Digest Algorithm. <http://www.faqs.org/rfcs/rfc1321.html>, 1992.
- [56] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
- [57] D. Rossi. Orchestrating Document-based Workflows with X-Folders. In *SAC '04: Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 503–507, New York, NY, USA, 2004. ACM Press.
- [58] R. Sandhu. Rationale for the RBAC96 Family of Access Control Models. In *RBAC '95: Proceedings of the 1st ACM Workshop on Role-based Access Control*, pages 1–8, New York, NY, USA, 1996. ACM Press.
- [59] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.
- [60] R. Sandhu and P. Samarati. Access Control: Principles and Practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.
- [61] Security Assertion Markup Language (SAML) (version 1.1). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security#samlv11. OASIS, 2004.

BIBLIOGRAPHY

- [62] Security Assertion Markup Language (SAML) (version 2.0). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security#samlv20. OASIS, 2005.
- [63] Security in a Web Services World: A Proposed Architecture and Roadmap. <http://www.ibm.com/developerworks/library/specification/ws-secmap/>. IBM, 2002.
- [64] J. Shen and S. Qing. A Dynamic Information Flow Model of Secure Systems. In *ASIACCS '07: Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*, pages 341–343, New York, NY, USA, 2007. ACM Press.
- [65] Shibboleth Project (version 1.3). <http://shibboleth.internet2.edu/>. Internet2, 2007.
- [66] R. Simon and M. E. Zurko. Separation of Duty in Role-based Environments. In *CSFW '97: Proceedings of the 10th IEEE Workshop on Computer Security Foundations*, pages 183–194, Washington, DC, USA, 1997. IEEE Computer Society.
- [67] L. Singaravelu, J. Wei, and C. Pu. A Secure Information Flow Architecture for Web Services. In *2008 IEEE International Conference on Services Computing*, pages 182–189, Los Alamitos, CA, USA, 2008. IEEE Computer Society.
- [68] SOAP (version 1.2). <http://www.w3.org/TR/soap12/>. W3C Recommendation, 2007.
- [69] M. Sohlenkamp, P. Mambrey, W. Prinz, L. Fuchs, A. Syri, U. Pankoke-Babatz, K. Klöckner, and S. Kolvenbach. Supporting the Distributed German Government with POLITeam. *Multimedia Tools Appl.*, 12(1):39–58, 2000.
- [70] W. Stallings. *Network and Internetwork Security: Principles and Practice*. Prentice Hall, 1995.
- [71] W. Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 2006.
- [72] D. Stinson. *Cryptography: Theory and Practice*. CRC, 1995.
- [73] E. Stohr and J. L. Zhao. Workflow Automation: Overview and Research Issues. *Information Systems Frontiers*, 3(3):281–296, 2001.
- [74] R. Summers. *Secure Computing: Threats and Safeguards*. McGraw-Hill, 1997.

BIBLIOGRAPHY

- [75] Universal Description, Discovery and Integration (UDDI) (version 3.0.2). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec. OASIS Standard, 2005.
- [76] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.
- [77] J. Warner and V. Atluri. Inter-Instance Authorization Constraints for Secure Workflow Management. In *SACMAT '06: Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, pages 190–199, New York, NY, USA, 2006. ACM Press.
- [78] J. Warner, V. Atluri, R. Mukkamala, and J. Vaidya. Using Semantics for Automatic Enforcement of Access Control Policies among Dynamic Coalitions. In *SACMAT '07: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, pages 235–244, New York, NY, USA, 2007. ACM Press.
- [79] Web Services Business Process Execution Language (WS-BPEL) (version 2.0). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel. OASIS Standard, 2007.
- [80] Web Services Description Language (WSDL) (version 2.0). <http://www.w3.org/TR/wsdl20/>. W3C Recommendation, 2007.
- [81] Web Services Federation Language (WS-Federation 1.1). <http://www.ibm.com/developerworks/library/specification/ws-fed/>. BEA Systems, BMC Software, CA, IBM, Layer 7 Technologies, Microsoft, Novell, VeriSign, 2006.
- [82] Web Services Policy (version 1.5). <http://www.w3.org/TR/ws-policy/>. W3C Recommendation, 2007.
- [83] Web Services Security (WS-Security 2004). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss. OASIS Standard Specification, 2006.
- [84] Wildlife Enforcement Monitoring System. <http://www.unu.edu/wems/>. UNU, 2008.

BIBLIOGRAPHY

- [85] Workflow Management Coalition – Terminology and Glossary. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf. Workflow Management Coalition, 1999.
- [86] Workflow Management Coalition – The Workflow Reference Model. <http://www.wfmc.org/standards/docs/tc003v11.pdf>. Workflow Management Coalition, 1995.
- [87] WS-SecureConversation (version 1.3). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-sx. OASIS Standard, 2007.
- [88] WS-SecurityPolicy (version 1.2). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-sx. OASIS Standard, 2007.
- [89] WS-Trust (version 1.3). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-sx. OASIS Standard, 2007.
- [90] Web Services Architecture. <http://www.w3.org/TR/ws-arch/>. W3C, 2004.
- [91] XML Encryption Syntax and Processing. <http://www.w3.org/TR/xmlenc-core/>. W3C, 2002.
- [92] XML Schema. <http://www.w3.org/XML/Schema>. W3C, 2001.
- [93] XML Signature Syntax and Processing. <http://www.w3.org/TR/xmldsig-core/>. W3C, 2002.