

X-Flow

- *A Secure Workflow System*

December 31st, 2005

Martin Strandbygaard Jensen (s001522)

Abstract

This masters thesis describes a model and prototype implementation for a secure workflow system, that facilitates creation of documents according to a workflow description, where the result of each activity is digitally signed.

The requirements of a document workflow is analyzed in the context of a workflow with 10-100 participants, and the security aspects of document creation as part of a workflow are investigated.

The practical barriers and opportunities of using digital signatures as a replacement of hand-written signatures, thereby allowing workflow support of processes with formal requirements for documentation, are also analyzed.

Based on the security and workflow requirements of such a system a supporting data model is developed. The model is implemented as an XML Schema specification and makes use of the XML Schema language to allow model checking to be performed in an XML parser.

Finally, a prototype application supporting this data model is implemented according to the specified model requirements.

Sammenfatning

Dette speciale beskriver analyse, design og implementering af et sikkert workflow system der kan anvendes til at udarbejde og godkende dokumenter, der skal dannes efter en bestemt proces, og som kræver formel godkendelse med bindende underskrift.

Baseret på en analyse af sikkerheden i et elektronisk workflow system, og en sammenligning med et manuelt workflow, opstilles en kravspecifikation for et system der er sikkert på de identificerede områder hvor et manuelt system ikke tilbyder nogen sikkerhed. Endvidere er systemet designet til at være lige så sikkert på øvrige områder.

Ud fra denne kravspecifikation udarbejdes et XML schema der definerer datamodellen for en (XML) baseret filcontainer, der kan bruges som "token" i et workflow. Containeren understøtter vilkårlige filformater og versionering af ændringer.

System omdeler automatisk en dokumentcontaineren mellem hver deltager i workflowet, ud fra en fast beskrivelse af workflowet gemt i containeren, Design og prototype understøtter alle simple transitioner defineret i [1] samt multi choice, og designet understøtter automatisk multi merge.

Rapporten omfatter også design og implementering af en prototype på et system der anvender den beskrevne datamodel.

Systemet er designet som en client-server arkitektur (begge dele programmeret i Java), med et tilhørende webinterface til udvalgte funktioner.

Preface

This thesis was prepared at Department of Informatics and Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring a Master of Science degree in engineering.

The work of this thesis was carried out over a period of 6 months, from July 1st, 2005 to December 31st, 2005, and was supervised by Robin Sharp from IMM, DTU.

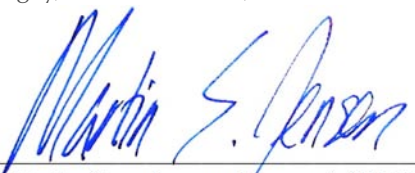
Acknowledgments

I would like to thank my supervisor Robin Sharp for taking an interest in this project and for our many and long discussions and certificates and digital signatures.

Also a great appreciation for the work my parents put into proof reading this report.

Finally, I would like thank my girlfriend, Karin, for her patience and support the past six months.

Lyngby, December 31st, 2005



Martin Strandbygaard Jensen (s001522)

Table of Contents

Abstract	I
Sammenfatning	II
Preface	III
1 Introduction	6
1.1 Problem Definition	7
1.2 Setting the Stage	8
1.3 Thesis Overview	8
1.3.1 Chapter Organization	9
2 Domain Description	10
3 Taxonomy of Current Systems	13
3.1 Workflow Systems	13
3.2 Competitive Analysis	15
3.2.1 Platform Requirements	17
3.2.2 Workflow Capabilities	18
3.2.3 Security	18
3.2.4 Price	18
3.2.5 Summary	19
4 Workflow Systems	20
4.1 Defining a Workflow	20
4.2 Transition Patterns	21
4.2.1 Simple Transition Patterns	22
4.2.2 Complex Patterns	24
4.3 Graphical Representation of Workflows	26
4.3.1 Petri Nets	27
4.3.2 Unified Modelling Language (UML)	27
4.3.3 Extending the Models	28
4.4 Document Workflow Support	28
4.4.1 Optimizing Workflow Specifications	29
4.4.2 Document Aging	29
4.4.3 Document Versioning	30
5 Security Analysis	31
5.1 Preconditions	31
5.2 Threat Agents	32
5.2.1 Organizational Process and Colluding Users	33
5.2.2 Platform Security	33
5.3 Threat Macros	34
5.3.1 Classification of Macros	35
5.4 Security Objectives	36
5.5 Comparison to Manual Workflows	38
6 Identification of Role	39
6.1 Applied Cryptography	40
6.1.1 Symmetric Encryption	40
6.1.2 Asymmetric Encryption	41
6.1.3 Public-Key Authentication	41
6.1.4 Digital Certificates	42
6.2 Legal Framework	43
6.2.1 Current Legislation and Market Adoption	44
6.3 Current PKI Models	45
6.3.1 Trusting a Certification Authority	45

6.4	Summary	46
7	Requirements Capture	47
7.1	Use Case Analysis	47
7.2	Workflow	48
7.2.1	Workflow Activities	48
7.2.2	Process Support	49
7.2.3	Signature Scope and Ordering	50
7.2.4	Error handling	50
7.2.5	Workflow Specification	51
7.2.6	Workflow Administration	52
7.3	Security	53
7.3.1	Trust Model	53
7.3.2	User Identification (Authentication and Signature)	53
7.3.3	Access Control (Authorization)	54
7.3.4	System Availability	54
7.3.5	Document Confidentiality	55
7.3.6	System Auditing	55
7.3.7	Version Control	56
7.4	System Architecture	56
7.4.1	Document Support	56
7.4.2	Platform Support	57
8	Container Specification	59
8.1	Container Format	59
8.1.1	Versioning Model	60
8.2	Specification Format	60
8.3	Data Model Design	61
8.3.1	Workflow Support	61
8.3.2	Verification Protocol	61
8.3.3	XML Signature	61
8.4	XML Schema Design	62
8.4.1	General Structure	62
8.4.2	Common Schema Elements	63
8.4.3	Server Endpoint (<info>)	64
8.4.4	Workflow Specification (<workflow>)	65
8.4.5	Document Versions (<document>)	68
8.4.6	Signature Protocol (<transactionlog>)	69
8.4.7	Controlling Attribute and Element Values	71
8.4.8	Schema Namespaces	71
8.4.9	Sealing the Container Structure	71
8.5	Creating a Container Instance	72
8.5.1	Creating a Workflow Specification	73
8.5.2	Adding the Initial <receipt> Element	73
8.5.3	Adding a Subsequent <receipt> Element	74
9	Secure Workflow Model	75
9.1	System Architecture	75
9.1.1	Client Interface	75
9.1.2	Client-Server Communication	76
9.1.3	Application Server	76
9.1.4	Database	77
9.2	XML Programming Model	77
9.3	User Interface	77
10	Model Implementation	78
10.1	Accessing the Container	78
10.1.1	Parser Configuration	78
10.1.2	Loading A Container (Client)	79
10.2	Workflow Engine	79
10.2.1	Selecting the Current Step (Client)	79
10.2.2	Verifying the Current Step (Server)	80
10.2.3	Multi choice and -merge Container (Server)	80
10.3	Certificate Access	80

10.3.1 Certificate Device Plugin Architecture	81
10.4 Signing and Verification	82
10.4.1 Signing an Element	82
10.4.2 Verifying a Signature	82
10.5 Client-Server Communication	83
10.6 Audit Trail	83
10.7 Web Interface	84
11 Model Analysis	85
11.1 Workflow Support	85
11.1.1 Supported Transitions	85
11.1.2 Container Template Instance	86
11.2 Model Checking using XML Schema Language	86
11.2.1 Parsing Time	86
11.2.2 Memory Usage	87
11.3 Security Analysis	87
11.3.1 User Authentication	88
11.3.2 Enforcing Authorization	88
11.4 System Test	88
12 Conclusion	89
Bibliography	90
A Contents on CD-ROM	92
B Comparison of Security in Manual and Electronic Workflow Systems	93
C Use Case Diagrams	95
D The Client User Interface	96
D.1 Overview	96
D.2 Screen Elements	96
D.2.1 Common Objects	96
D.2.2 Info Tab	98
D.2.3 Reviewing Tab	98
D.2.4 Authoring Tab	99
E The Web Interface	100
E.1 Overview	100
E.1.1 Standards Compliance	100
E.1.2 Security	101
E.1.3 External Configuration	102
E.2 User Interface	103
E.2.1 /login.php	104
E.2.2 /changepassword.php	104
E.2.3 /logout.php	104
E.2.4 /list.php	104
E.3 Administrator Interface	105
E.3.1 Managing Users	105
E.3.2 Managing Roles	107
E.3.3 Managing Workflows	109
F Admin Tool	111
G System Test	112
G.1 Test Data	112
G.1.1 Sequential Process	112
G.1.2 Unit Testing	113
G.2 Test Results	114
G.2.1 Unit Test	114
G.2.2 Functional Test	114
H Source Code	117
Index	118

List of Figures

1.3.1 Structure of this thesis	9
2.0.1 Simplified use case for document modification	11
4.1.1 Components comprising a generic workflow model	21
4.2.1 Sequence Pattern	23
4.2.2 Parallel Split Pattern	23
4.2.3 Synchronization Pattern	23
4.2.4 Exclusive Choice Pattern	24
4.2.5 Simple Merge Pattern	24
4.2.6 Simplification using Complex Transition Patterns	26
4.3.1 Sequence Pattern as a Petri Net	27
4.3.2 Simple transition patterns as Petri Nets	27
4.3.3 Sequence Pattern as a UML state chart	27
4.3.4 Simple transition patterns as UML state charts	28
8.4.1 Definition of metaDataType	63
8.4.2 Definition of the Info Element	64
8.4.3 High Level Model of a Workflow	65
8.4.4 Definition of nextStepGroup Element	66
8.4.5 Definition of a step Element	67
8.4.6 Definition of nextStepGroup Element	67
8.4.7 Definition of a Document Element	69
8.4.8 Definition of the Transactionlog Element	69
8.4.9 Definition of a Receipt Element	70
10.1. Sequence Diagram - Client Loading Container	79
10.3. UML class model of client certificate support	81
10.3. Selecting a certificate device	82
C.0.1 Complete Use Case for Retrieving a Document	95
C.0.2 Complete Use Case for Submitting a Document	95
D.2.1 GUI - Common Fields	97
D.2.2 GUI - Container Information	98
D.2.3 GUI - Reviewing a Document	99
D.2.4 GUI - Modifying a Document	99
E.2.1 Sitemap - Users Access	103
E.3.1 Sitemap - Managing Users	105
E.3.2 Sitemap - Managing Roles	107
E.3.3 Sitemap - Managing Workflows	109
G.1.1 UML diagram of sequential process	113
G.1.2 System internal model of workflow in G.1.1	113

List of Tables

3.1.1 Common Specific Workflow Applications	15
3.2.2 Summary of Capabilities	19
4.2.1 Complex Transition Patterns	25
5.2.1 Threat Agents	33
5.3.1 Threat Macros	35
5.3.2 Classification of Threat Macros	36
5.4.1 Security Objectives	37
11.1. Feature Matrix	85
11.2. Parsing Time	86
11.3. Security Level	88
B.0.1 Security in Manual Workflow System	94
G.2.1 Test Results	116

1

Introduction

Current business applications offer extensive support for automated processing of many types of transactions involving formally specified and structured content according to established business processes in organizations employing those systems. Specifically, processing of most financial transactions, such as purchase orders and invoices, is very well supported by a multitude of financial systems and enterprise resource planning systems (ERP).

In organizations employing ERP or similar systems, no human interaction is necessary except specification and approval, and this has led to great improvements in the efficiency of business administration.

In contrast to this strong support for controlled and automated processing of formally specified and structured content, few organizations employ automated systems for processing informal and unstructured content. This type of content includes all manners of electronic documents, e.g. business strategies and sales forecasts, that are created and communicated within an organization.

Many documents are created according to a predefined process, and regulatory, industry, or organizational requirements often mandate documentation of process execution. Generally, this is achieved by maintaining a paper trail of signatures that denote intermediate process decisions. However, this approach is quickly becoming an obstacle to efficient process execution, as well as limiting the usability of the process documentation. Furthermore, numerous corporate scandals of late, have increased the necessity of documenting process execution and formally establishing responsibility within the process. One example being the Sarbanes Oxley act [2], passed in response to the Enron- and World-Com scandals, affecting all companies, or subsidiaries thereof, registered with the Securities and Exchange Commission (SEC) in the US.

The Sarbanes Oxley act, among other things, requires companies to implement internal controls addressing key processes, and the execution of these controls must be documented.

Until recently there was no legal or technological replacement for the manual processes involved in signing documents. However, within the past 5 years several countries (including Denmark [3]) have passed legislation that makes certain types of electronic signatures legally binding. In March 2003 the first OCES certificates were issued in Denmark, and in 2005 its position as an electronic equivalent to the handwritten signature was confirmed in a report by The Danish Ministry of Justice [4].

The increasing amounts of information and more stringent formal process requirements has created a need for systems, that automate aspects of document creation and approval. This thesis investigates the problem of automated processing of *content*, utilizing digital certificates to create irrefutable and legally binding documentation of process execution.

A note on terminology

Before the introduction of electronic storage, a strong bond existed between information and the storage medium on which this information was captured, e.g. news was printed in the newspaper. Thus a document was a well defined entity, as was the relationship between a document and information; *a document contained information or data*.

With the advent of electronic storage, meta data, and structured formats (e.g. XML), this relationship became gradually blurred, and one might argue that it no longer exists; certainly modern development paradigms have shown separation of content and presentation, to be a superior strategy in development of WEB applications.

In this thesis the following terminology is employed

1. *Content* can be anything that can be communicated electronically. Thus no characteristics can be inferred about the content.
2. *Meta data* describes *content*.
3. *Content* and describing *meta data* is *information*.
4. A *document* is a visual *representation of information*. Thus a *document* may be an instance of *content* and *meta data*, in which the meta data also describes presentation.

This definition lends itself to *XML* terminology, however, the special definition of a document is necessary, to properly express those situations in which a document may not be separated in *content* and describing *meta data*.

It should be noted, that the definition is directional in that a *document* is *always content*, whereas *content* not necessarily is a *document*.

The system described in this thesis assumes documents, hence in the following this term will generally be used.

1.1 Problem Definition

The aim of this thesis is to develop a prototype for a software system, that supports secure document handling for a wide variety of document formats in the context of a workflow system. Specifically, the system should be useful as a secure workflow system for processing electronic documents.

The system will enable easy handling by multiple participants of electronic documents that must

1. Be created according to a formal process (workflow)
2. Ensure irrefutable commitment of all participating roles (signatures)
 - (a) with legal significance (assuming the external requirements are available)
3. Provide traceability in the document creation process
4. Facilitate easy documentation of the above properties

In so doing, the system must

- be easy to use
- allow specification of flexible workflows
- allow distribution of arbitrary document formats
- function on many different platforms
- use digital certificates for authentication of participants
- distribute documents between participants

1.2 Setting the Stage

This thesis presents the specification, design, implementation, and evaluation of the *X-flow Secure Document Workflow System*; a collaboration system in which users create, modify, and review documents. Documents are created according to a formal process specification, all modifications are digitally signed, and a document contains complete traceability of versions, user activities, and signatures, and it enforces access control.

Several systems or applications address the problem of coordinating document collaboration in a workflow, and they can be divided into several different categories according to their approach to solving this problem. Some systems focus not only on document collaboration but on coordinating tasks of any kind (e.g. an XML broker in a middle-ware application), while other systems in turn address the problem of processing very specific kinds of documents (e.g. controlling the source code files of a computer program).

The *X-flow* system focuses on processing documents that *must* be formally approved and signed by any subset of the involved users, and where the approval process must satisfy legal requirements and be able to replace existing manual processes.

The *X-flow* system takes a document centric approach in which *everything* the system knows about a document at a given point in time, is represented by the corresponding state of the document. Specifically, the formal security properties the *X-flow* system ensures, *must not* depend on the system or on any data stored within, rather the formal properties *must* be proven by the document itself.

1.3 Thesis Overview

The remainder of this report is structured in the following way.

- Chapter 1 presents an introduction to the topic of this thesis and the scope of the problem that is addressed.
- Chapter 2 presents a domain description of the intended workflow system. The domain description covers the process mechanism of the system, a description of how the system is used by a user and an administrator, and a description of the intended environment in which the system will operate.
- Chapter 3 is a taxonomy of current systems and how they compare to the domain description in chapter 2. The chapter describes different types of document workflow systems currently being used and analyzes a number of representative systems based on selected characteristics.
- Chapter 4 gives an introduction to workflow modelling, including simple and complex elements that can be used to describe a workflow. This chapter also states the general workflow objectives required by a document workflow system, in which a document is created according to a predefined process (see section 1.1-1).
- Chapter 5 analyzes the detailed security requirements of a document workflow system, and based on this analysis a number of security objectives are expressed, which are required to ensure security of a document as defined in the security analysis. (see section 1.1-2.*)
- Chapter 6 gives a general introduction to electronic identification and signatures based on cryptography (e.g. digital signatures) and gives an overview of how current legislation allows digital signatures to be used as replacements for traditional handwritten signatures. Finally, this chapter presents an overview of current standards governing digital signatures and their market adoption.
- Chapter 7 summarizes the objectives stated in chapter 4 and 5 and restates them as specific system requirements. These system requirements are combined with use case derived system requirements.

- Chapter 8 outlines the design for a prototype system that implements the system requirements described in chapter 7.
- Chapter 9 describes the data model used by the prototype system.
- Chapter 10 describes how each part in the system is implemented including key algorithms for signing, validation, and workflow processing.
- Chapter 11 analyses the implemented prototype, in terms of (1) the requirements actually implemented by the prototype, and (2) of how major design choices affects the overall performance of the prototype (both in terms of system requirements and actual performance).
- Chapter 12 concludes the work of this thesis and summarizes the designed model, the prototype, and the performance of the chosen design.

1.3.1 Chapter Organization

The final system design and prototype model are developed through a series of steps, each step using input from previous steps, as illustrated in figure 1.3.1.

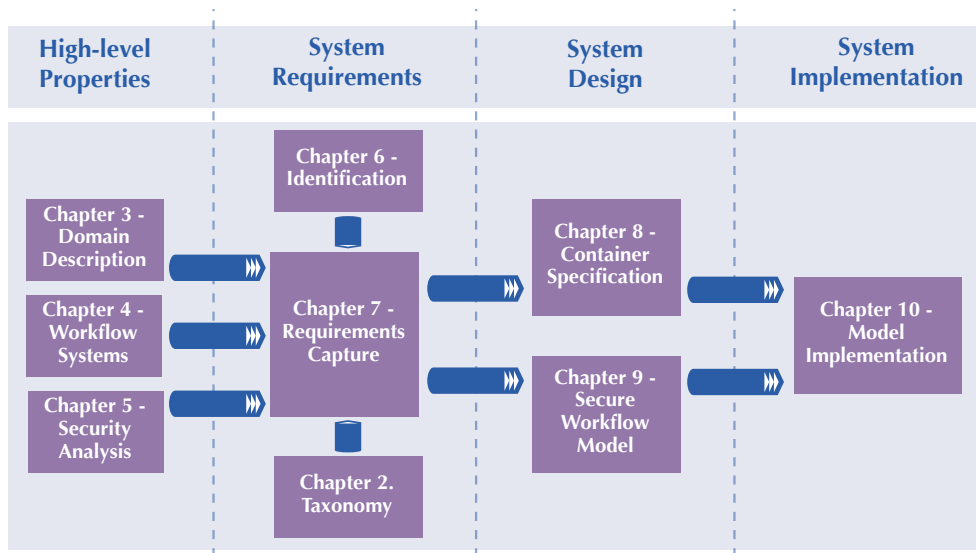


Figure 1.3.1 The figure shows how the individual chapters relate in terms of what is covered in each chapter, and how they combine in the overall design process.

2

Domain Description

This chapter is an extended description of the *X-flow* system described in section 1.2, and provides an informal description of the use and purpose of the intended workflow system.

The description defines the functional boundaries of the system, and identifies the (functional) sub-components. This includes a detailed description of the intended functionality, relevant deliberations, and specific choices that impacts the final system implementation. Based on this description it is possible to identify high level use case scenarios for the system, and major classes of the final design. Detailed use case scenarios are discussed in section 7.1, and its class model in chapter 10.

X-flow is at the same time a document specification, a workflow specification, and a client-server system. Users access the *X-flow* system, from which they can retrieve documents they must edit. When a user is done editing the document, the user uploads the document to the server that in turn distributes the document to the next recipient according to a predefined workflow specification.

A user commits to the changes by digitally signing the document, and the *X-flow* system enforces that the document is signed, and that the changes the user has made correspond to those in the workflow specification.

The Process

The system is used to control the creation of a **single document (production data)** through a **workflow process**, which means that a single document is created through input from **multiple participants (roles)**.

A workflow process is an existing business process, through which a single document is created and approved. To express a workflow process, a business process must be identified and mapped, both in terms of order of activities and participating roles. This mapping is not facilitated by the system and should be prepared before using the system.

The system need not support very complex workflows. The system need only support typically occurring business processes that are actually documented, and very detailed and complex processes are extremely difficult to model correctly, hence relatively simple workflows will normally be the order of the day. Specifically, the system will only support one active editor regardless of whether the document editing system of choice supports multiple editors¹.

The system is intended to enable processes that are bound by formal requirements (regulatory etc.) in which the execution of each activity must be formally approved and have a legal signif-

¹Several text editors such SubEthaEdit use the Zeroconf protocol for concurrent editing

icance. Since the system is executing (and enforcing) the process, the system, must also be able to handle process exceptions caused by external factors such as human errors.

The System

The system allows the roles to contribute in **arbitrary order (workflow process)**, but the workflow process is specified **prior** to starting the creation of a document, thus when the document creation is started, the possible paths of the process are known. The participants themselves cannot specify the workflow process in which participants contribute to the document, but the participant starting the creation of a document, may choose from a number of **established** processes.

The document creation order, specifies both the actual order (workflow process), but also **what actions (activities)** each role may take on the document.

The system controls the process of document creation, which means that the system itself isn't used to actually create the document. Rather the system provides a **container** for an arbitrary document, and this container is used for document distribution within the workflow.

This abstraction ensures that the system doesn't mandate that documents be created in a specific way, and that the domain of document creation is separated from the domain of the creation process. This also means that workflow **components** should be accessible to any application that implements the correct interface (API) to the workflow component.

Documents are edited in an external application, thus the system must allow roles to add and extract documents stored within the distributed container, while at the same time ensuring that a role only performs activities they are authorized to perform. This is done by

1. only continuing the workflow process if the activity taken by a role was **authorized**
2. only promoting the (container) submitted by a role as the **current master** if the container is valid

This means that the system must verify that (1) holds for all activities of all previous roles.

A workflow has ended when the chosen process requires no more activities.

Using the System

A role only needs limited interaction with the system. A role interacts with the system, when the role **retrieves** a container in order to perform an activity, and when the role **returns** the container to the system after having completed the designated activities. This high-level use case scenario is illustrated in figure 2.0.1.

When a role access the system in order to add or retrieve a container, the role must **log in** to the system by **authenticating** as a role **known** to the system.

Any role using the system will have some form of organizational capacity, hence the system identifies a role by searching in directories of known organizational capacities (company address book, centralized user directory etc.). Thus the system itself will not facilitate user management, but will provide an interface to access **external** user identities.

A role may have more than one document waiting, and when a role isn't editing a document, the documents are held on a **central server**. The role accesses the system through a **single window**

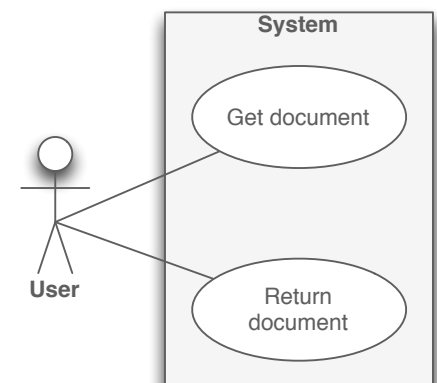


Figure 2.0.1 Simplified use case for document modification

(web page) from which the role has access to **status information** about all waiting documents. From this window, a role can perform all relevant activities pertaining to the awaiting documents. However, actual document editing is performed locally (not on the server).

To edit a document, a role first retrieves the container in which the document is stored and then proceeds² to extract the document from the container for local editing. This step is necessary and important, because a role must be able to validate a document locally and not have to rely on information presented by the system.

Creating a Document

Not all roles can **create** all documents. A workflow process corresponds to a business process, hence only the originator in the business process can create a document. A role can create a new document from any of the container templates available to that role, and only templates in which a given role is originator, is available to that role.

When starting a workflow, a role simply creates a document and adds this to a new container instance intended for the type of workflow the role is initiating. Consecutive roles then proceed to retrieve the container, edit the document, and return the container containing the edited document.

A workflow not only comprises sequential activities, but it can also involve simultaneous activities, and the workflow process definition may be dependent on the activities of some roles.

As a role edits a document outside the document container, the system needs to be available on the platform on which the role edits the document.

Administrating the System

The system does not require any on-going administration³ once it has been configured. Users can initialize workflows, and the users themselves are responsible for completing the workflows.

The system itself does not provide any means of *defining* a new workflow, but provides a specification from which workflows can be built.

The System's Environment

The system is intended to be used by a limited number of roles (10-100), since very few business processes incorporate large number of roles.

²The extraction process is easily wrapped so a user doesn't have to start multiple programs to actually retrieve the document.

³ This does not address normal administrative duties such as backup, dba, patching software etc.

3

Taxonomy of Current Systems

This section provides an overview of a number of existing workflow systems. It gives an introduction to the different types of systems that are normally used for workflow processing and compares the capabilities of a number of representative systems.

The systems are compared in terms of

- Price
- Workflow capabilities
- Security measures
- Platform requirements

These are the major features that should be considered when selecting a workflow system, and they are all relevant parameters to consider when answering the questions of (1) whether the intended system (already) exists, and (2) what barriers of implementation exists.

3.1 Workflow Systems

Workflow systems can broadly be divided into two groups

- developed as generalized workflow systems
- designed for a specific application domain

This categorization is relevant as many workflow systems are in fact varying types of content management systems extended with workflow support.

Specific workflow systems are generally self contained applications that require little or no customization¹, whereas generalized workflow systems typically need to be extended through customization.

¹Though they often require plenty of configuration to fit organizational models.

General Workflow Systems

Generalized workflow systems can further be divided into

- workflow engines
- self-contained applications

Workflow engines are system building blocks (programming libraries) used as drop-in functionality that can be used when developing new systems. A workflow engine in itself cannot actually be used to process any documents, rather it facilitates easy creation of such systems. Workflow engines generally relate to the problem of generic process automation, but all document workflow systems normally include some form of workflow engine as this architecture generally makes a system more extensible. Some workflow engines comprise an entire subsystem including e.g. user management and access control, while others only provide a workflow mechanism. As workflow engines are only used in conjunction with a development project, they will not be discussed any further.

Self-contained applications are normally referred to as middle-ware applications, because they route messages in between other systems. They have evolved from the application domain of transaction processing systems such as Microsoft BizTalk and IBM MQ Series into more generalized systems, that focus on processing the business process itself and make fewer assumptions about the environment in which they're used.

Specialized Workflow Applications

The most common types of systems include [5]

Version Control Is well known from the problem of source code management, where version control systems are used to manage different versions or releases of a program. A version control system is mainly used to capture different versions of a specific document (e.g. source code file) and will normally include some form of access control system.

Content Management System (CMS) First became known as a platform from which a website is published. As websites have evolved so have content management systems, and by now they're commonly used for controlling collaborative workflows that takes place across a website (e.g. a registration process). A content management system normally includes user management and access control mechanism, either by implementing its own security subsystem or by using that of the host system.

Document Management System Is often implemented as *version control for office documents*, and indeed many software developers also use their version control systems for this purpose. A document management system generally implements a simple sequence workflow, which ensures that only one current version can exist. It is also often used by public offices that must journal all correspondence. A document management system normally implements access control through the host system.

E-mail Is not a document workflow system per se, given that it has no means for *enforcing* document flow. However, from a practical standpoint e-mail systems cannot be ignored, given that e-mail is probably the single most used system for document collaboration. The original e-mail doesn't implement any access control mechanism, but can be extended with S/MIME that implements security through X.509 certificates.

Groupware Systems Generally extends point-to-point e-mail to better suit communication within a group, typically through bulletin boards and calendars.

Table 3.1.1 is a list of common systems from each of the five categories

Table 3.1.1: Common Specific Workflow Applications	
Version Control	BitKeeper Concurrent Versions System (CVS) Subversion (SVN) Visual Source Safe (VSS)
Content Management (CMS)	Apache Lenya Sitecore Zope
Document Management	Adobe Document Services cBrain EMC Documentum Fælles Elektronisk Sags- og Dokumenthåndtering (FESDH) Microsoft Sharepoint Scan-Jour
Groupware	Lotus Notes Microsoft Outlook phpCollab

Table 3.1.1 The table lists a selection of common workflow applications, grouped by type.

3.2 Competitive Analysis

The *X-flow* system must have a competitive advantage compared to existing systems. If several other systems already can do what *X-flow* is intended to do, and do it at a lower price, there seems to be little reason for creating *X-flow*.

In reality, few systems offer the exact same feature set making a direct comparison impossible, and the extensive impact of features such as platform support means, that even seemingly minor differences in feature set may justify a large difference in price.

If a company already uses SAP, implementing the SAP workflow component may be cheaper than implementing Domino Workflow, which also requires client licenses. In this scenario the added complexity of introducing another system [Lotus] is a hard to quantify factor that would probably impact work efficiency, hence also justifying the higher price tag of SAP.

Selecting Comparative Systems

The systems listed in table 3.1.1 are widely used, but as they are very domain specific, not all come close to the application domain of *X-flow*. The general workflow support of *X-flow* means that comparative applications should be selected from systems offering generic workflow support. The selection should also comprise systems that enjoy widespread adoption among the potential users (and offers a workflow component) as companies prefer to limit the number of applications in their portfolio.

Table 3.2.1 is a comparison of several common workflow systems.

Table 3.2.1: Comparison of Common BPM- and Document Management Systems

Vendor / Application	Price	Platform	Workflow Express-ability	Authentication Mechanism	Authorization Model	Logging / Auditing	Document Format Support
Adobe Document Services	\$\$\$\$	Linux, Mac, Solaris, Windows	Single step SEQ and rules processing. (Can be extended programatically to support all simple transition patterns.	Windows Authentication, X.509	RBAC, ACL	Container signature, log file	Single file (arbitrary format) in PDF-based container
Cosa BPM	-	Unix, Windows	All simple transition patterns	-	-	Log file	
Domino Workflow	\$\$\$	Mac, Unix, Windows	All simple transition patterns	LDAP, Notes ID file, Windows Authentication	RBAC, ACL	Container signature, log file	Arbitrary
EMC Documentum	\$\$\$\$	Windows	-	Simple, Windows Authentication	RBAC, ACL, Windows	Log file	Arbitrary
Microsoft Sharepoint	\$	Windows	Single step SEQ. (Can be extended programatically to support all simple transition patterns.	Windows	RBAC, ACL, Windows	(Document signature), log file	Arbitrary (full feature set is only available with MS Office formats, and arbitrary data included as objects).
Pallas Athena FLOWer	-	Windows	All simple transition patterns	N/A	N/A	Log file	Arbitrary
SAP R/3 Workflow	\$\$\$\$	Unix, Windows	All simple transition patterns	Simple, Windows Authentication, X.509	RBAC, ACL	Log file	Arbitrary
Tibco Staffware	\$\$\$\$	Windows	All simple transition patterns	N/A	N/A	Log file	Arbitrary
FileNet Visual WorkFlo	-	Windows	All simple transition patterns	Simple, Windows Authentication, X.509	RBAC, ACL	Log file	Arbitrary

Comparing the systems is done as follows

1. No company is about to throw out major investments in systems and applications, as well as accumulated experience, just to adopt a system with a limited number of users, and a very specific use. Hence, the first criteria is that a system offers acceptable platform support. This means that the impact to existing infrastructure should be limited, and preferably not require additional investments in supporting applications (and user education).
2. When an acceptable subset has been selected in terms of platform requirements, a company will evaluate whether the system supports the internal process it is intended to automate; a corollary to this is that no manager is going to select a system that is going to (negatively) influence his position (which a reorganization may cause). Regardless, it should be possible to customize the system to the organization, not the other way around.
3. The security requirements of the system will only be considered when platform requirements and workflow support has been satisfied. If those requirements cannot be met, the system is going to be too expensive to implement anyway, and the company may as well stay with the current manual processes.
4. The price is both the first and the last parameter to be considered, and essentially the price is considered in each of the three previous steps. The system is measured against the gain of implementing the system, which is another way of saying that the system is implemented if it is worthwhile.

3.2.1 Platform Requirements

The platform requirements comprise

- platforms on which the system can run
- document formats supported by the system

Generally platform requirements are a compromise between support for existing systems, while not limiting future evolution of infrastructure.

Operating Platform

The platform referred to in table 3.2.1 is the *client* platform; all companies can be expected to be able to operate servers running commonly available operating systems², however the client platform cannot be as readily changed.

Given the current widespread adoption of Windows, it should be assumed (and probably required), that a client is available for this platform, and indeed all systems offer this support.

Half of the systems offer clients for other platforms than Windows. If only Linux is supported, this is stated specifically, and if support spans all common Unix variants, support is referred to as Unix.

None of the systems use clients based on byte code languages such as Java or .Net, which could potentially provide more generic platform support, though rich clients implemented in a platform independent byte code language tend to be platform dependent anyway (e.g. [19]).

Document Formats

From a short term real life perspective, for most companies this is a matter of whether support for documents other than Microsoft Office formats is required. However, the use of PDF as the industry standard read only document format means that most companies would require support for this format as well. Realistically, few companies can do without at least limited support for

²Disregarding systems such as OS/400 and zOS that wouldn't be considered primary targets for such a system anyway.

arbitrary document formats.

All systems offer full support for arbitrary document formats. Some systems are based on a container format (Adobe and Microsoft), and this seems to be a desirable approach as it offers perceptive equality between signature and content being signed (e.g. users do not have to understand that *one* signature covers multiple files).

3.2.2 Workflow Capabilities

Realistically, few companies will require very detailed workflow expressability. To actually enable some form of workflow, a system must both support other transition patterns than the sequence pattern, and it must support many consecutive steps.

The systems fall into two categories: Those with almost no workflow support, and those with very detailed support. Those with limited support all implement a workflow model intended towards the perhaps most common workflow: The manager that must approve something written by a subordinate. However, this expressability is insufficient to support many frequently occurring business processes. The rest of the systems offer workflow support far more detailed than what will generally be required.

Most (large) companies will also require some way of integrating the workflow system with other systems, e.g. to facilitate automatic storage in an archive, once the document has been processed. All systems expose API's that can be used to integrate the systems with other systems.

3.2.3 Security

Currently, most companies *will* accept a new system that

1. only allows authentication using username and password
2. doesn't integrate with existing security infrastructures (LDAP, AD etc.)

Also, most companies view *greater than normal* security features as a sure way of generating work for the it-support function. However, to meet the requirement of legally valid signatures, systems need to support X.509 certificates.

Companies will mostly be concerned with support for integrating authentication and authorization with Windows, as just about all companies use this on clients (and certainly above the level of first line manager).

In terms of security model, authentication, and authorization all systems offer very good support. They all support RBAC and ACL with users being defined in a number of different directories, most support user authentication using X.509, and all systems integrate with Windows security.

While the systems do support X.509 for authentication, few support X.509 for signing documents, and those that do, only implement a very simple signature model without integrating signature with the workflow execution.

3.2.4 Price

As previously stated, the price is a factor that is considered in each step of the selection process, and a system will always be chosen to *fit the bill*.

Ideally, the system will be chosen based on a cost-benefit analysis of the gains in terms of optimization, however, these gains are hard to quantify and are affected by other factors such as existing infrastructure.

Regardless, workflow support at the document level is a relatively new technology, and doesn't enjoy widespread adoption. This makes it a relatively expensive technology to implement.

3.2.5 Summary

Table 3.2.2 summarizes the comparison of the capabilities of existing systems.

Table 3.2.2: Summary of Capabilities	
Platform Requirements	Acceptable support for multiple operating platforms. All systems support arbitrary document formats.
Workflow Capabilities	Either very limited (Adobe, EMC, Microsoft) or very extensive (the rest) support.
Security	Good support for access control using X.509 certificates, mainly because Windows support this natively and most applications can use Windows security. Limited support (Adobe and Microsoft) for signing documents. No support for integrating signature into the workflow, and no properties can be assigned to a signature.
Price	Regardless of relative pricing, all systems are very expensive. This is fairly new technology enjoying limited market adoption, hence the extra premium.

Table 3.2.2 The table summarizes the capabilities of existing workflow systems.

Generally, no single system addresses the problem that *X-flow* is intended to solve. Several systems offer extensive workflow support, and the Adobe solution offers limited signature capabilities. However, none of the systems integrate document signing into the decision process inherent in the workflow, and none of the systems provide a systematic framework for attaching properties to a signature.

4

Workflow Systems

To build a system that facilitates managed document flow, or workflows, it is necessary to develop a formal model of the classes of workflows the system will support. This chapter gives a general introduction to workflow systems. It defines a workflow, and provides a taxonomy of elements that can be used to express very complex workflows.

Section 4.4 states the requirements of a document workflow system that constitutes a fair compromise between expressability and complexity of implementation, and which the developed system should support. Chapter 7 expands on these requirements, and in chapter 3 the selected subset is compared to the (workflow) capabilities of current workflow systems.

4.1 Defining a Workflow

A workflow is a model expressing the work that must be performed in a given business process, or simply a process. The creation of a document from a functional (or business) point of view, is normally referred to as a *process* or *business process*. Designing a system that automates this process can be likened to capturing this process as a model and implementing this model, and the implementation will then express the flow of work that constitutes the captured process.

In the following, the terms and description of a workflow stated in [1] (and [20]) will be used to describe a workflow. In [1] an arbitrary workflow is decomposed into the following components:

Workflow Process Definition expresses the *activities* that must be performed, in what *order* they should be performed, and whether they all need to be performed (e.g. through branch *transitions*). A workflow process definition is generic, and is said to be instantiated in a specific case (e.g. when referring to the processing of a specific document). In the following, a *workflow process definition* will be referred to as a workflow.

Routing/Control Flow refers to the *order* in which the *activities* comprising a *workflow* must be performed

Thread Of Execution Control is relevant when parallel *activities* may occur.

Control Data is meta data used to describe the *workflow* and the routing within (e.g. conditionals on branch statements).

Activity is an *atomic* unit of work that must be performed as part of the *workflow* (any activity that needn't be atomic is itself a new *workflow*).

Transition defines how to move between activities. The types of transitions that are allowed determines the complexity of *workflow* that may be expressed, and the ease with which complex *workflows* may be expressed. Many complex *transitions* may be modelled from simpler *transitions*.

Role is the person performing an *activity* as part of the *workflow process definition*. The word *role* is used to express that this need not be a specific person (e.g. it may be an organizational function).

Application is used by the *role* to perform the *activity*

Production Data are the modifications to the document that are performed as part of the workflow.

Figure 4.1.1 shows a graphical representation of a simple workflow and illustrates the different components of a workflow.

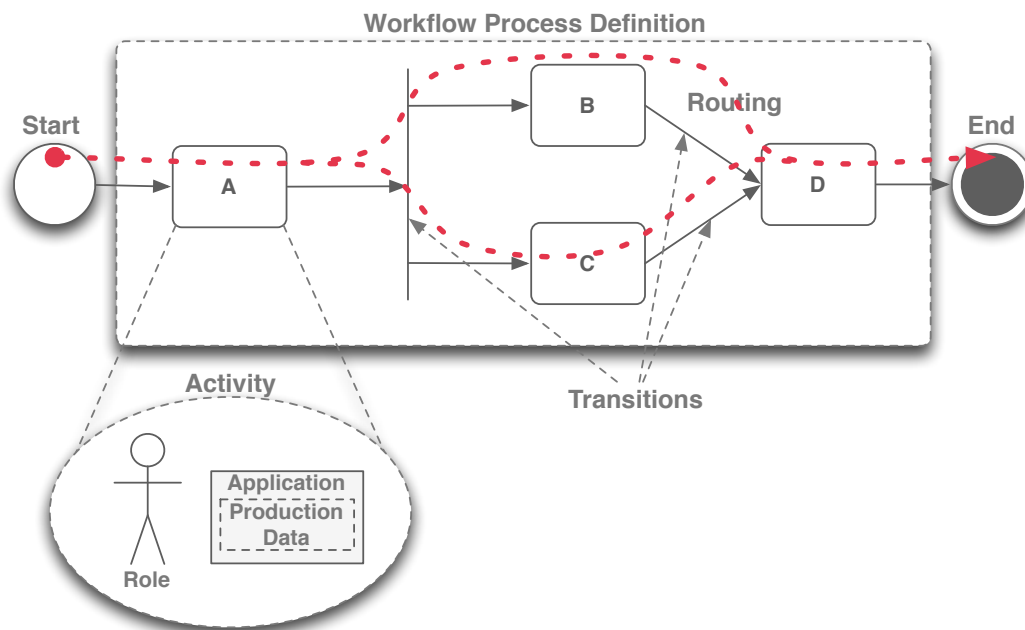


Figure 4.1.1 Components comprising a generic workflow model

The workflow starts with a *sequence transition* from the start state into an activity, and the break-out shows the elements comprising a generic activity. It is common to indicate start- and end-states using labeled circles as is shown here. From activity A, the workflow branches into two threads of execution through *parallel split*, visiting activities B and C respectively, and finally the execution threads are merged using through a *simple merge* pattern visiting activity D and completing execution in the end-state.

The description illustrates a *generic* transition, hence this description has no impact on the complexity of the workflows that can be described, meaning that arbitrary workflows can be described.

4.2 Transition Patterns

As noted in the definition of transitions in workflows, a transition is simply a rule governing how to move between two or more activities, and as such a large number of transitions can be described. However, most transitions may be expressed from a limited and well defined set of transition patterns. This is also a requirement for formally specifying the workflows that a system should support.

Transition patterns can relate to many different application domains, e.g. ([21], [22], [23])

- Business Process
- Flow control
- Resource allocation
- Case handling
- Exception handling
- Transaction management

The scope of this thesis is limited to transition patterns that address flow control.

Much effort has gone into cataloguing transition patterns within each application domain, and e.g. the search criteria `business workflow pattern` yields 57 results on Amazon.com . In [1] 21 transition patterns for flow control are identified, and this definition is used as basis for the model developed in this thesis¹. The patterns are divided into

- Simple Patterns
- Complex Patterns

Simple patterns are entities that cannot be expressed by means of other (simple) patterns, while complex patterns in most cases may be expressed by simple patterns [1]. However, using the notation of complex transition patterns will greatly simplify the expression of many workflows, and allow easier analysis, than would have been possible using only transition patterns.

4.2.1 Simple Transition Patterns

This section provides an overview of the five simple transition patterns defined in [1].

- Sequence
- Parallel Split
- Synchronization
- Exclusive Choice (XOR)
- Simple Merge

The transition patterns also correspond to the basic control flow constructs defined by the Workflow Management Coalition (WfMC) [24].

Each pattern is illustrated with a small figure showing (possible) pre- and post activities. The illustrations use the same notation as figure 4.1.1. Each illustration is accompanied with a short description of the pattern's function, and an example of how this pattern may relate to a document workflow system. Section 4.3 introduces other graphical notations for workflows.

4.2.1.1 Sequence

The sequence pattern is the simplest of all transition patterns, and occurs in most workflows. The pattern expresses a transition from one activity to another, in which just one pre- and post activity exists, and where the transition is bound by no conditional.

Connecting two activities by a sequence transition as is shown in figure 4.2.1 means that activity B is enabled in a workflow, as soon as the work in activity A is completed.

¹This work currently appears to be the authoritative voice on the topic. A search on `workflow pattern` on Citeseer show that Will M.P. van der Aalst (author of [1] is either the author of, or cited by 45 out of 57 results

This transition pattern is used to model a situation where a document is moved from one person to another, e.g. where one person has written a report, and another person has to confirm the calculations.

Any workflow that involves exactly

- one input
- one output
- one concurrent document editor²
- one concurrent activity



Figure 4.2.1 Sequence Pattern

may be modelled using just this one transition pattern [24]. However, processing in a workflow based on only the *sequence* transition scales linearly with number of activities in the workflow, which is not very inefficient.

E.g. if a proposal for a new standard must be approved by each member of a large committee, the *entire* process stops if just one member is unable to perform his assigned activity.

4.2.1.2 Parallel Split (AND-split)

The parallel split pattern introduces concurrency in a workflow by introducing an AND-split. In figure 4.2.2 this means that both B and C will be enabled when A completes.

In a parallel split the current thread of execution *will* split in two or more concurrent threads of execution. No conditionals apply to the transition. A parallel split is assumed to result in at most two threads, and when a parallel split is encountered it will *always* split in two. Without this restriction the meaning of a parallel split may be ambiguous.

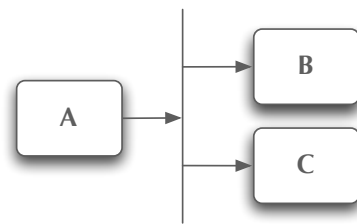


Figure 4.2.2 Parallel Split Pattern

If a parallel split was allowed in the example from 4.2.1.1 it would solve the problem of the process halting on just one member. If a parallel split was allowed (and probably some complementary merge pattern), then even if one member of the committee failed to approve the proposal, the rest would still be able to do so, and the person in charge of the process could concentrate on getting the last approval.

4.2.1.3 Synchronization (AND-join)

Synchronization only makes sense, if a parallel split is allowed. The synchronization pattern is used as barrier synchronization of two concurrent threads of execution (that are merged into one thread during the transition).

In figure 4.2.3 this means that if A completes before B, the thread executing A cannot progress until B has also completed.

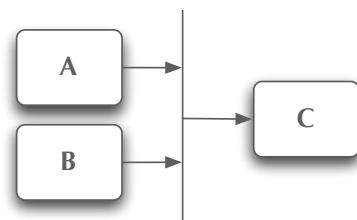


Figure 4.2.3 Synchronization Pattern

Building upon the example of 4.2.1.2, a synchronization transition would be the obvious way to rejoin the multiple threads of execution. If the proposal required the approval of each member of the committee before further processing, the synchronization pattern could be used to enforce this.

²defined as a person editing the actual document

4.2.1.4 Exclusive Choice (XOR-split)

The exclusive choice transition pattern implements an XOR-split. Conditionals must be attached to each transition out of the preactivity, and

$$B \neq C$$

must always hold.

When A completes (figure 4.2.4), a conditional, c , must be specified that decides if B or C should be enabled. The conditional must always evaluate to a single activity, and if $c \equiv B$ it follows that C can never be enabled (given the simple workflow in the figure).

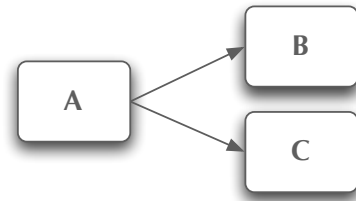


Figure 4.2.4 Exclusive Choice Pattern

An exclusive choice would be the expected transition out of the committee's deliberation on the proposed standard. Either the proposal is accepted, and the standard is adopted, or it is rejected.

4.2.1.5 Simple Merge (OR-join)

The simple merge transition pattern is used to join one or more *alternative* branches in a workflow, meaning that if this pattern is employed it is assumed that the two branches being merged, are never executed in parallel. In figure 4.2.5 this means that in a workflow execution where A is enabled, it is assumed, that B will not be enabled during the same execution.

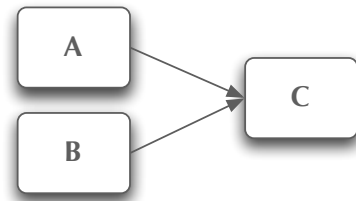


Figure 4.2.5 Simple Merge Pattern

The other case is the complex transition patterns *multi merge* described in section 4.2.2, where the two branches are executed in parallel, and A and B complete in $t_1 \neq t_2$.

A simple merge transition would be the final transition of the proposal from the previous example. Regardless of the result from the exclusive choice from section 4.2.1.4, the committee would in the end be expected to finish the processing of the proposal through some form of official statement coupled to the proposal.

4.2.2 Complex Patterns

This section provides an overview of more advanced branching and synchronization patterns, which are referred to as complex transition patterns. State based patterns are not treated.

Complex transition patterns express commonly occurring patterns that can only be expressed using a number of combined simple transition patterns, and replacing such a construct with a single pattern reduces the overall complexity of a workflow diagram. The complex transition patterns are, however, as the name implies more complex to implement, and even most commercial workflow systems only select a small subset of the patterns [1].

Table 4.2.1 lists the complex transition patterns described in [1] and provides a brief description of each, and the section concludes with a combined example using a number of complex transition patterns to show the power of their expressability.

Table 4.2.1: Complex Transition Patterns [1]	
Pattern	Description
Multi-choice	Based on a decision a <i>number</i> of branches are chosen (OR-split)
Synchronizing Merge	Multiple paths converge, and one or more are synchronized
Multi-merge	Same as above but without synchronization, hence following activities are activated once for every branch that is merged.
Discriminator	Merges several branches, but only activates following activities once, for the first incoming branch. Later incoming branches are ignored.
Arbitrary Cycles	A point where an activity may be performed an <i>arbitrary</i> number of times.
Implicit Termination	Simply terminates subprocesses when no more activities remain.
Multiple Instances Without Synchronization	For one process and activity is enabled multiple times.
Multiple Instances With A Priori Design Time Knowledge	An activity is enabled multiple times, and the number of times is known at design time.
Multiple Instances With A Priori Run-Time Knowledge	An activity is enabled multiple times, and the number of times is known at some point during run-time.
Multiple Instances Without A Priori Run-Time Knowledge	An activity is enabled multiple times, and the number of times is neither known during design- or run-time.
Deferred Choice	A point where one of several branches is chosen, but the decision is not based on data or decision, rather a number of choices is presented, and once one is chosen, the others are implicitly withdrawn.
Interleaved Parallel Routing	A number of activities are executed in an arbitrary order, the order being decided at run-time, and only one activity is active at a given moment.
Milestone	An activity is enabled if a given <i>state</i> of the workflow has been reached.
Cancel Activity	An enabled activity is disabled.
Cancel Case	Removes an entire workflow instance.

Table 4.2.1 Complex transition patterns described in [1]. The list only includes patterns for branching and synchronization.

The expressability of complex transition patterns can be shown using a simple example. Suppose a document has been created and reviewed using a sequence pattern, and that the document must now be reviewed by a larger audience, e.g. 16 members of a technical committee. Expressing that using simple transition patterns (for the moment assuming that all members actually complete their review), this would require 7 `AND-splits` just to branch 16 times. However, branching using `AND-splits` we must also join using (7) `AND-joins`, which in turn *requires* that all committee members have completed their review, before the workflow execution can proceed. That is 14 transitions just to branch and join synchronously, with even more if the case where not all committee members complete their review must be handled.

However, the same scenario can be expressed using just one `multi choice` to branch, and one `synchronizing merge` to join the branches. Figure 4.2.6 shows this reduction with 8 branches

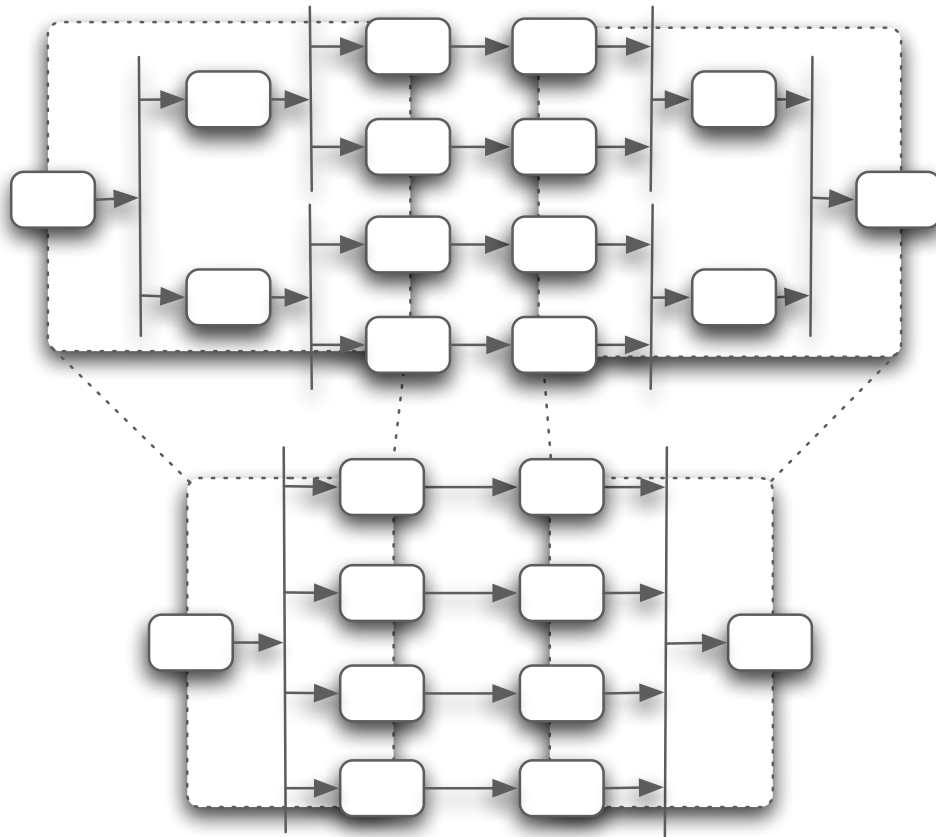


Figure 4.2.6 The figure shows how a multi choice and synchronizing merge can be used to simplify workflow specification.

4.3 Graphical Representation of Workflows

This section describes how Petri Net and Unified Modelling Language (UML) activity diagrams may be used to model the simple transition patterns defined in section 4.2.1.

The informal graphical notation used to introduce transition patterns in section 4.2 was intended to convey the relationship between the graphical construction and the functional purpose of each pattern. However, a formalized notation (process modelling technique) is necessary to enable systematic analysis of workflow models.

Numerous formal and informal graphical modelling techniques (graphical modelling languages) have been developed since the sixties [20], each offering different advantages in certain contexts. Some techniques are intended for generalized modelling of systems, whereas other techniques are developed to model specific classes of systems.

The diversity of graphical modelling languages has been further expanded by the fact that few application implementations are alike. When creating a computer application that implements a given graphical modelling language, compromises are made that yield a system supporting a subset of the original language, and often also supporting constructs that the original language did not. This being a common problem when implementing a standard of any kind.

Both Petri Nets and UML activity diagrams have a limited vocabulary making it easy to provide a correct implementation, and both languages can be used to formally analyze a model.

4.3.1 Petri Nets

Petri Nets are used extensively as a modelling tool both in scientific research, and in planning and optimizing commercial processes.

The transition patterns described in section 4.2.1 may be expressed as Petri Nets, allowing formal analysis of workflow semantics. This section provides a Petri Net model corresponding to each of the 5 transition patterns described in section 4.2.1. The Petri Nets in this section uses named transitions.

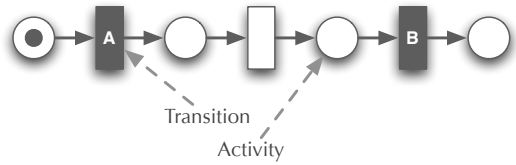


Figure 4.3.1 A sequence pattern represented as a Petri Net diagram

Figure 4.3.2 shows a Petri Net representation of each of the simple transition patterns. The more verbose notation results in more complex figures than those used in section 4.2.1.

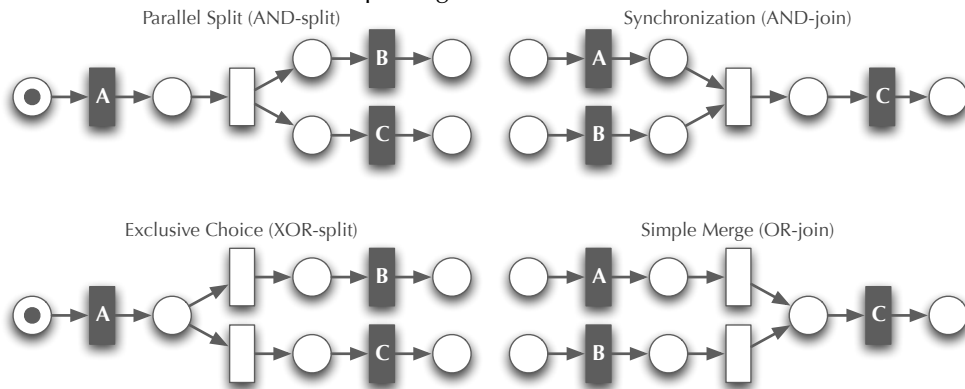


Figure 4.3.2 Petri Net representation of simple transition patterns

4.3.2 Unified Modelling Language (UML)

The Unified Modelling Language standard is controlled by the Object Management Group (OMG), and as of version 2 comprises 13 different graphical models (diagrams) [25], of which state diagrams and activity diagrams are commonly used for workflow modelling.

The use of state diagrams for modelling workflows stems from UML version 1, in which activity diagrams was viewed as a special case of state diagrams. In UML version 2 the notation of activity diagrams has been formalized and in the following only activity diagrams will be described.

Figure 4.3.3 shows a sequence transition pattern with labelled *activity* and *transition* elements. The notation of activity diagrams is very similar to that used in section 4.2.1, but UML activity diagrams also introduce other elements (most notably the *object* element not shown here).

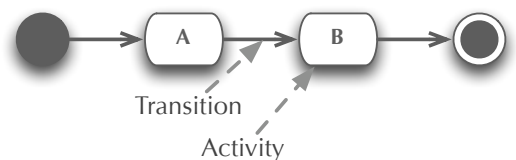


Figure 4.3.3 A sequence pattern represented as a UML state chart

Figure 4.3.4 shows the UML representation of each of the remaining 4 simple transition patterns.

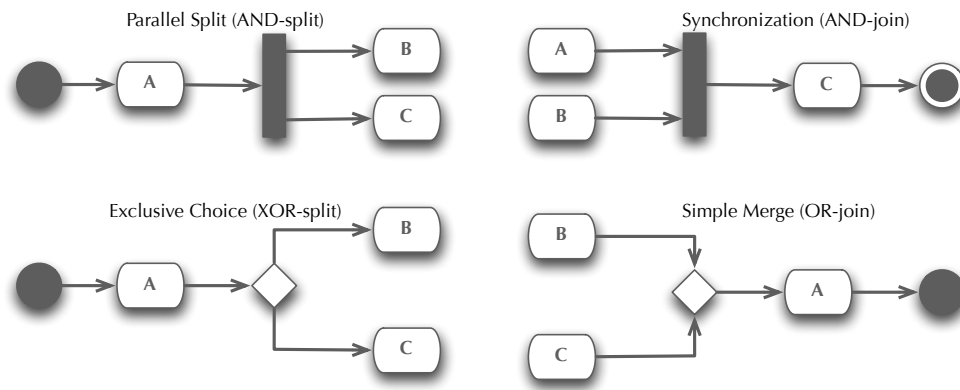


Figure 4.3.4 UML representation of simple transition patterns

UML activity diagrams may be used as model checker by [26][27]

- verifying properties of the activity diagram itself, and
- verifying other (UML) models (diagrams) against an activity diagram

Several UML based CASE (computer aided software engineering) tools incorporate some form of model checking (e.g. IBM Rational Software Development Platform [28], Enterprise Architect [29], and others [30][31]), as well as offering both forward- and reverse engineering between code and UML model.

In the rest of this thesis, UML activity diagrams will be used to express workflow models.

4.3.3 Extending the Models

Both Petri Nets and UML activity diagrams can express simple transition patterns, but several other elements of a workflow, as defined in section 4.1 are not readily apparent. Specifically neither notation denote role to the activities and flow of production data.

In UML, object constraint language (OCL) [32] may be used to condition transitions, and by extension may be used to express role. This addition is necessary in formal analysis of models, but is ill-suited for more informal visual model analysis.

The common approach to graphically representing roles is to partition the model in "swim-lanes", where each lane represents a distinct role [20], and this notation will also be used here.

Production data may be expressed by the `object` element that is defined for UML activity diagrams.

4.4 Document Workflow Support

The possible ways a document can be distributed between recipients are determined by the supported transition patterns, hence the *X-flow* system need only support transition patterns typically required by a document workflow. The *X-flow* system should

- allow specification of workflow (4.4-1)
- automate the workflow process (4.4-2)
- support simple transition patterns (4.4-3)
- have a well defined decision process (4.4-4)

Specifying a workflow is a complex task that most companies have designated employees perform, and the inflexibility, that a user cannot specify a workflow at will, means that specifications must be relatively static, hence a workflow must be

- well understood
- well defined
- static

By correlation, in most situations complex transition patterns are not necessary because

- users don't understand the process
- business processes that would require them are too difficult to describe
 - and such complex processes would change frequently, as things become done differently
- workflows incorporating them become too error prone

Complex or large workflow specifications are more prone to changes due to external factors (re-organizations, doing things differently, etc.) making them less usable. If a user needs to create a document, and the workflow system is *unavailable* due to lack of workflow specification, the user will simply resort to other means, e.g. e-mail.

Disregarding scalability most workflows can be realized using the sequence pattern. AND-patterns improve scalability by allowing concurrent processing, and OR-patterns by reducing the number of steps in the workflow.

4.4.1 Optimizing Workflow Specifications

One pattern that occurs frequently when many people collaborate to create a document, is the case where one person has authored a document that *must* be approved by $m > 2$, $m \cap n$ number of people, before the author can continue working on the document. This pattern can be realized using nested AND-split, but assuming that transitions cannot be cascaded, this construction will introduce a number of redundant activities.

A better approach would be to introduce a pattern that allows the current thread to branch into $t > 2$ threads, of which $0 < x \leq t$ must be executed. This would only require one transition to model the scenario, and would not halt the workflow, if a number of the branches were not visited. Likewise, a pattern is required that will merge this type of branch.

Summarizing, the *X-flow* system should support:

- Multi-choice transition pattern (4.4-5)
- Synchronizing merge transition pattern (4.4-6)

4.4.2 Document Aging

In a document workflow system, multiple different versions of the original document is created during the course of workflow execution which necessitates that a rank of the document versions is established.

In a workflow comprising only sequence transitions ranking documents is not a problem, as a newer versions directly precede their ancestors, but in a workflow with branches several different versions of a document may exist at any given time.

In the *X-flow* system, document aging should be performed according to the following rules:

- concurrent modification of a document is not possible (4.4-7)
- a modification activity creates a new version of a document (4.4-8)
- a new version precedes all current and previous modifications and reviews (4.4-9)
- at any given time only *one* version of a document is valid (4.4-10)

The workflow models covered in this chapter assume one start point and one end point, and the result of a workflow that results in multiple versions in the end point is ambiguous, hence multiple concurrent versions are disallowed.

The aging scheme does not prevent splits in a workflow, but it does prevent splits where modification occurs in one or more of the branches. A split, where modification only takes place in one branch, does *not* create multiple versions, as only a modification creates a new version, and is disallowed because of rule 4.4-9. As this case does not create multiple versions, it could be allowed without creating undefined states in the workflow, but negative impact on implementation and data model outweighs the advantages that this branch optimization might carry.

Using the above mentioned rules ensures that when a modification activity is entered, there will be no review in a part of the workflow graph, that this modification activity does not precede or is ancestor of, which means that the backing data model does not have to support several branches. On the other hand, allowing reviews to take place on different part of the workflow graph, would in some cases provide a nice optimization, where work can continue on a document, while formal approval is obtained e.g. for a project milestone draft.

4.4.3 Document Versioning

Content creation that takes place over a number of activities can be created by:

- Modify and delete
- Create and Append

In the first scheme, modifications are directly applied to the result of a previous activity, while in the second, the result is *appended* to the result of the previous activity. The latter is the principle of version control systems, as it maintains a copy of every version that has existed.

The intended workflow system should be self documenting, hence

- the workflow system should provide document versioning (4.4-11)

Keeping a copy of all versions that have been created, ensures a complete transaction log of the workflow execution, which is also a required security objective of the system³

³Keeping a copy of all versions is also necessary in order to be able to verify a signature applied to a specific document version.

5

Security Analysis

System security should be based on an analysis of the threats the system will face. A system may have many weaknesses but if no corresponding threats exist, they have no impact on the overall system security.

This chapter presents a security analysis of a secure workflow system. It introduces the threats the *X-flow* system will likely face, and based on this, relevant security objectives of the system are defined. Chapter 7 decomposes these objectives into specific system security requirements.

Objectives that ensure that the *workflow* is *executed* correctly are not covered in this chapter, although they may be described using properties of security objectives. Workflow objectives are described in section 7.2, where the security objectives are modified and extended to comprise workflow objectives.

The security analysis is organized as follows:

1. Section 5.1 states the preconditions that apply to the *X-flow* system. A system cannot just be *secure*, and defining necessary prerequisites and external requirements is fundamental to expressing what security objectives the system *should* ensure.
2. Threats may be directed towards a system from many directions, and section 5.2 defines what agents may represent threats towards the system. The section also describes the threat agents' relations to the system, and the reason why certain threat agents are irrelevant as stated in section 5.1.
3. Based on the defined threat agents, section 5.3 states the possible threat macros that relate to the system. A threat macro constitutes a threat agent, the targeted security metric, and a description of the applicable scenario. The section also provides a qualitative description of reason for the identified threat macros.
4. Finally section 5.4 states the required security objectives of the system, in terms of the defined threat macros and system security policy.

5.1 Preconditions

The *X-flow* will not be designed to counter any conceivable threat, hence a number of assumptions about the system operating environment and implementation details are required.

In this analysis, as well as the rest of this thesis, physical security and operating system platform security will not be treated, nor will their impact on system security. Thus, if an attacker gains one of the following

- physical access to the hardware on which the system is running

(5.1-1)

- privileged access to the operating system platform on which the system is running (5.1-2)

it will be assumed that system security is compromised. Conversely,

- physical security and operating system security are assumed (5.1-3)

Any system that is used by *users*, requires a degree of user administration, or identity management (IdM), either by providing its own administration interface or by accessing an existing directory of users. Assigning a set of user credentials to the correct person is fundamental to maintaining system security, and it will be assumed that

- A process for assigning a set of user credentials to the correct person is in place, and (5.1-4)
- This process is always effective (5.1-5)

Generally, systems are designed to be more resilient towards external threats, than the threat from an authenticated user with malicious intent. It will be assumed that

- Any *single* user may have malicious intent (5.1-6)
- Two users will not conspire to subvert system security (5.1-7)

Hence employing the system in an organization with adequate separation of duty will prevent users from misusing the system, and the final system should be adequately robust, so that

- one person (internal or external) acting alone, cannot compromise system security (5.1-8)

This also means that system security should *not* rely on security controls external to the system¹, e.g. network access controls (packet filters, authenticating proxies, etc.).

Even though operating system security is ensured, other code executed on the same platform may render the system insecure, hence it is assumed that

- All executed code works correctly (5.1-9)
- No malicious code is executed on the platform, that hosts any part of the workflow system (5.1-10)

5.2 Threat Agents

A threat agent constitute an entity representing one or more threats towards a target system, and comprise the *direction* and *type* of threat in terms of the analyzed system. The *direction* of a threat may either be *internal (I)* or *external (E)* in relation to a system, where a system is defined as

- Governing organizational processes
- Involved people (users, system administrators, etc.)
- Code that is executed as part of running the system

Internal threats are those posed by any part of the system (using the above definition), whereas external threats are directed from the surrounding environment towards the system. Type of threat denotes which part of the system or its surroundings that pose the threat.

Using direction and type as properties of a threat agent, the *X-flow* system faces the following threat agents²

¹Physical- and operating system security being exceptions

²Threat types that are italicized are excluded from the analysis according to section 5.1

Table 5.2.1: Threat Agents						
Label	Direction	Type	System State			
5.2-1	Internal	Workflow decision process				
5.2-2		User administration process				
5.2-3		System administration process				
5.2-4		Normal user		Authenticated	Authorized	
5.2-5				Unauthenticated	Unauthorized	
5.2-6		Privileged user		Authenticated	Authorized	
5.2-7				Unauthenticated	Unauthorized	
5.2-8		Company employee		Unauthenticated		
5.2-9						System administrator
5.2-10						Executed code
5.2-11	External					
5.2-12				Provider of infrastructure		
5.2-13				Competitor		
5.2-14				Process stakeholder		

Table 5.2.1 The table lists identified threat agents

From the table it should be apparent, that a simple classification of direction based on organizational relationship isn't possible. A workflow system will be employed in an organization, but it isn't given that all users will *belong* to that organization, however, all users are viewed as part of the system.

Consequently, *external* agents largely comprise agents, that have no affiliation with the system, but may still pose a threat, such as malicious code that is targeted at an entirely different system, but still affects the system. E.g. when a virus inhibits network access, the target system becomes the network, but a workflow system would also be affected.

There are actually more possible venues of attack directed from within the system itself, than are posed externally, suggesting that very verbose and explicit security controls addressed towards internal threats are necessary.

5.2.1 Organizational Process and Colluding Users

Given the preconditions of section 5.1 possible threat agents represented by processes for system support (5.2-2) and maintenance (5.2-3) are not addressed in the resulting security objectives, and neither are threats relating to a collusion of users. While some systems do offer robustness against possible threats from such agents, e.g. using secure hardware, they're extremely costly, and most still require supporting organizational processes to maintain a given security level.

Compared to other systems with special or above average security requirements, this scope is in line with adopted *best practice*. Systems for processing electronic medical records (EPJ) being implemented in most hospitals are not required to be robust against failures of internal procedures, and no explicit requirements for secure hardware apply to EPJ systems [33].

5.2.2 Platform Security

Disregarding systems used by militaries, few systems are required to be, nor are they designed to be

- resilient towards (security) failures by subcomponents, such as errors in software (5.2.2-1)
- robust from a breakdown in surrounding security controls (5.2.2-2)

Consequently, the stated security objectives of this analysis will not consider the effects of agents relating to (5.2.2-1) and (5.2.2-2) identified by (5.2-9) and (5.2-10/11), as stated in (5.1-9) and (5.1-9).

Software development is starting to be able to provide an avenue for mitigating the impact of (5.2.2-1)³, as is new processor features such as (hardware) memory protection through XOR'ing the write and execution bits⁴. Despite these advances, few applications can guarantee security if the underlying platform security is circumvented through malicious code (e.g. a virus or trojan horse). Likewise, if the security of system dependancies (network, data interfaces, etc.) are compromised, the level of security will at least experience a serious degradation.

Agents representing malicious code generally target client systems, as the less controlled operating environment of client systems make them more susceptible to e.g. viruses or trojans. Especially online banking systems targeted towards private persons face the problem of maintaining system security with a user base of disparate clients, but the generic problem of malicious code in all organizations, shows that the client susceptibility should be considered a universal problem.

While few systems are completely resilient towards the impact from malicious code, many systems are designed to guard subcomponents, especially storage of authentication credentials. Many payroll systems require users to authenticate using FIPS 140-3 compliant hardware tokens, Danske Bank provide web banking access using an ActiveCard one-time-password generator (OTP), and several VPN systems also use OTPs, e.g. RSA SecureID.

Hence, security objectives shouldn't completely disregard the effect of malicious code, rather security objectives should only address the impact of malicious code with respect to authentication data.

5.3 Threat Macros

The *X-flow* system is intended to be used as described in section 2 in a typical business or public organization. No risk analysis can accurately express the environment of all secure workflow systems, as threat agents are dependent on the specific system, hence this analysis expresses an average threat level. Organizations that face specially resourceful adversaries, operate in a different environment than the one assumed here.

Each threat is expressed as a threat macro comprising the threat agent, and a description of the scenario. In the scenario descriptions it is assumed, that *Alice*, *Bob*, and *Carol* are the participating roles in a workflow, and *Mallory* is not part of the workflow⁵, and each macro is linked to a corresponding threat agent from table 5.2.1.

Security threats are normally expressed in terms of one or more of the following metrics [36], and these will be used for grouping the macros:

1. Authentication
2. Authorization
3. Availability
4. Confidentiality
5. Integrity

Of the metrics, 3 - 5 are the ones most commonly used in a security analysis from a business perspective, e.g. an internal audit of IT-security, and several [business oriented] standards are

³E.g. OpenBSD implements a number of technologies to prevent buffer overflows[34], and process separation in OpenBSD (chroot) or FreeBSD (jail) prevents global system access of rogue processes, and virtualization in Sun Solaris (containers) [35], VmWare, or Xen takes this concept even further.

⁴A feature that has gained mainstream adoption through Windows XP on supporting processors

⁵As is the accepted way of informally expressing security threats

structured in terms of these three metrics ([37][38]), though more theoretical work in IT-security also takes this approach [39].

The following table lists the threat macros stated in terms of all five metrics.

Table 5.3.1: Threat Macros			
Label	Metric	Agent	Scenario
5.3-1	Authentication	5.2-8 - 5.2-11, 5.2-13 - 5.2-14	<i>Mallory</i> tries to access the system as <i>Alice</i>
5.3-2		5.2-4 - 5.2-7	<i>Alice</i> tries to access the system as <i>Bob</i>
5.3-3		5.2-8 - 5.2-11, 5.2-13 - 5.2-14	<i>Mallory</i> submits a document as <i>Alice</i>
5.3-4		5.2-4 - 5.2-7	<i>Alice</i> submits a document as <i>Bob</i>
5.3-5	Authorization	5.2-8 - 5.2-11, 5.2-13 - 5.2-14	<i>Mallory</i> obtains a document from the system
5.3-6		5.2-4 - 5.2-7	<i>Alice</i> obtains a document intended for <i>Bob</i>
5.3-7			<i>Alice</i> sends a document to <i>Carol</i> for which <i>Bob</i> was the intended recipient
5.3-8		5.2-8 - 5.2-11, 5.2-13 - 5.2-14	<i>Mallory</i> submits a document that <i>Alice</i> has previously submitted
5.3-9		5.2-4 - 5.2-7	<i>Alice</i> submits a document that she has previously submitted
5.3-10			<i>Bob</i> submits a document that <i>Alice</i> has already submitted
5.3-11	Availability	5.2-3, 5.2-8 - 5.2-14	<i>Mallory</i> makes the system unavailable to <i>Alice</i>
5.3-12		5.2-4 - 5.2-7	<i>Alice</i> makes the system unavailable to only <i>Bob</i>
5.3-13		5.2-3, 5.2-8 - 5.2-14	<i>Mallory</i> makes the system unavailable to <i>Alice</i> , <i>Bob</i> , and <i>Carol</i>
5.3-14		5.2-4 - 5.2-7	<i>Alice</i> makes the system unavailable to <i>Bob</i> and <i>Carol</i>
5.3-15	Confidentiality	5.2-2 - 5.2-3, 5.2-8 - 5.2-9, 5.2-12 - 5.2-14	<i>Mallory</i> reads a document from the system
5.3-16		5.2-4 - 5.2-7	<i>Alice</i> reads a document intended for <i>Bob</i>
5.3-17			<i>Alice</i> reads a document previously intended for <i>Bob</i>
5.3-18			<i>Alice</i> learns what documents are waiting for <i>Bob</i>
5.3-19		5.2-2 - 5.2-3, 5.2-8 - 5.2-9, 5.2-12 - 5.2-14	<i>Mallory</i> obtains information about the execution of a workflow in the system
5.3-20	Integrity	5.2-8 - 5.2-14	<i>Mallory</i> changes a document waiting for <i>Alice</i>
5.3-21		5.2-4 - 5.2-7	<i>Alice</i> changes a document outside the scope of a workflow activity assigned to her
5.3-22		5.2-8 - 5.2-14	<i>Mallory</i> changes a document <i>Alice</i> has previously changed
5.3-23		5.2-4 - 5.2-7	<i>Alice</i> changes a document <i>Bob</i> has previously changed

Table 5.3.1 The table lists the compiled threat macros for the system

Normally, threat macros would be created in terms of an established classification of information security, e.g. it is *assumed*, that *Mallory* is disallowed to read a document, but in many cases this need not be the case. As it is, the threat macros are based on a *default deny* policy.

5.3.1 Classification of Macros

The macros presented in table 5.3.1 correspond to known classes of attacks on information security. Using the attack classification adopted by W. Stallings [40] and C. Pfleeger [39] the following grouping of macros is obtained

Table 5.3.2: Classification of Threat Macros				
Type	Attack Class	Threat Metric	Description	Corresponding Threat Macros
Passive	Interception	Confidentiality	Access information sent between two communicating parties	5.3-15 - 5.3-19
Active	Impersonation	Authentication, Authorization	Pretend to be another <i>valid</i> user	5.3-1 - 5.3-4
	Interruption	Availability	Prevent two parties from communicating with each other	5.3-11 - 5.3-14
	Fabrication	Authentication, Authorization	Create information and send it to one or more communicating parties	5.3-3 - 5.3-4
	Modification	Integrity	Modify information in transit between two communicating parties. This attack is often referred to as man-in-the-middle (MITM)	5.3-20 - 5.3-23
	Replay	Authentication, Authorization	Resend intercepted information to the system	5.3-7 - 5.3-10

Table 5.3.2 The table groups the threat macros according to the security metric to which they relate.

This grouping implies that most attacks are directed towards mechanisms that enforce authentication and authorization, and correspondingly these security metrics will be given more attention in stating the security objectives of the system.

5.4 Security Objectives

The security objectives state how the system counters the security threats described in section 5.3, thus ensuring that the formal properties required of a workflow cannot be compromised.

In principle, the security objectives can be expressed as the inversion of the identified security threats, however this approach is not optimal as it

- may be ambiguous
- does not address accepted risks
- does not address *unknown* security threats⁶

The first two points are apparent when considering the security objective complementary to threat macros 5.3-11-14, which would state that *the system must be available*. This objective is *ambiguous* as available isn't defined, and certainly most all organizations accept some degree of unavailability.

The following security objectives are defined in terms of a *default deny* policy, and address the security threats of table 5.3.1. The security objectives counter the listed threat macros, and are expanded to clarify possible ambiguities (the expansions are italicized).

In terms of all five metrics, the system should satisfy the following security objectives:

⁶It is only possible to state known threat macros, but security objectives may be employed to eliminate the impact of new threat macros.

Table 5.4.1: Security Objectives			
Label	Metric	Objective	Relating Threat
5.4-2	Authentication	A role <i>must</i> be authenticated to access the system	5.3-1 - 5.3-2,
5.4-3		A role <i>must</i> be authenticated to perform an activity	5.3-3 - 5.3-4
5.4-3a		<i>A role need not be authenticated to modify a document</i>	
5.4-3b		A role <i>must</i> be authenticated to commit the result of an activity	5.3-1 - 5.3-2
5.4-4		<i>A role must be authenticated to access production data</i>	
5.4-5		<i>A role must be authenticated to access control data</i>	
5.4-6	Authorization	<i>When a role must be authenticated, a role must also be authorized</i>	5.3-7 - 5.3-10
5.4-6a		A role <i>must</i> be authorized to commit the result of an activity	5.3-5 - 5.3-6
5.4-7		A role <i>must</i> be authorized to access production data	
5.4-7a		<i>A role must be separately authorized to access production data of separate activities</i>	
5.4-8		<i>A role must be authorized to access control data</i>	
5.4-8a		<i>A role must be separately authorized to access control data of separate activities</i>	
5.4-9	Availability	<i>The system must not limit a role's organizational capacity</i>	5.3-11 - 5.3-14
5.4-10	Confidentiality	Production data <i>must only</i> be accessible to authenticated and authorized roles	5.3-15 - 5.3-19
5.4-11		<i>Control data must only be accessible to authenticated and authorized roles</i>	
5.4-12	Integrity	<i>An activity must not be refutable</i>	5.3-21
5.4-13		<i>Only an activity can change control data</i>	5.3-20, 5.3-22 - 5.3-23
5.4-14		<i>Only an activity can change production data</i>	

Table 5.4.1 The table lists the stated security objectives for the system

The objectives state that all access and interaction with the system requires explicit authentication and authorization and so does access to both production- and control data. Objective 5.4-2a simply states that authentication is not implemented in terms of the document, which means that if someone were to obtain a document, they may be able to modify it. As this has little significance as long as the document cannot be submitted into a workflow, this relaxation makes for a much simpler implementation.

5.5 Comparison to Manual Workflows

This section compares a manual workflow process with a system that satisfies the stated objectives. A new system should provide at least the same level of security (where relevant), as the old system, be it manual or automated, it is replacing.

The comparison is based on the threat macros listed in table 5.3.1, and the following table summarizes the relative security using either a paper based or an electronic document workflow system⁷. Each system is graded in terms of its resilience, where **green** is very resilient, and **red** is least resilient.

Table 5.5.1: Comparison																							
	5.3.1-																						
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Manual	Green	Red	Green	Yellow	Green	Red	Red	Green	Yellow	Yellow	Green	Green	Green	Green	Green	Red	Red	Red	Green	Green	Yellow	Green	Yellow
Automated	Green	Green	Green	Green	Green	Green	Green	Yellow	Green	Red	Yellow	Yellow	Red	Yellow	Red	Yellow	Green	Yellow	Green	Green	Green	Green	Green

An electronic system doesn't appear to rate much better than a manual system (electronic:manual - **1:2**, **3:2**). However, labels 11 - 14 all address system availability and disregarding these the ratios improve (**1:6**, **1:1**). Thus, an electronic system can be at least as secure as a manual system, and disregarding availability, an electronic system can be much more secure in terms of number of macros to which the system is not resilient.

⁷Appendix B provides a detailed comparison of the resilience of each system towards each threat macro.

6

Identification of Role

In this chapter the problem of correctly identifying and authenticating a role is analyzed, as well as how the system can support digital signatures.

In chapter 4 a role was defined as a person participating in a process, and this role/person was assumed ideal in the sense that the role would implicitly perform its capacity correctly. Section 7.3.1 established a trust model for a workflow, in which an activity performed by an identified and authenticated role is valid. Hence authentication and identification of roles becomes the foundation of the security within such a workflow system.

Authentication and identification are commonly treated synonymously as it is e.g. assumed that a certain user account on a system identifies a specific person who owns this account, but when the size (typically monetary) of transactions based on the trust of the identification increases, so does the requirements for the guarantees that the identification process offers. In short, we need to be *really* sure.

On the other hand authentication between two communicating parties is quite easy to establish even if their identities are completely unknown to each other, and several handshake protocols exist that will accomplish this.

Authentication and Signatures

The common approach to authentication is using a signature scheme. In the physical domain, a (physical) signature is the universal way of authenticating a person, and all official documents of certification (passport, drivers license) are provided with the signature of the holder of the document.

In principle, a signature is an identification of a specific person, that cannot be duplicated by any other person. Hence a signature uniquely identifies a person, and by extension guarantees that the person is whom the person purports to be.

Regardless of practical issues such as forgery, variations in reproduction of handwritten signatures, etc., this is the accepted property that is denoted a handwritten physical signature. However, it seems unlikely that a new signature scheme will be accepted, if it cannot offer any higher degree of assurance of identification than current physical signatures.

The (digital) signature scheme that is adopted for the system described here, must be able to *guarantee* the principle of a signature, and it must be possible to formally verify this guarantee.

6.1 Applied Cryptography

This section introduces the principles of using cryptography to provide confidentiality and authentication in IT-systems, and includes a description on how this is used to create a system of *digital signatures*.

6.1.1 Symmetric Encryption

Confidentiality is ensured through the process of encrypting data. Data is encrypted using a key, and depending on the chosen encryption algorithm, the same key will either decrypt the data, or it may require a different key. If the same key is used to encrypt and decrypt data, the process is called symmetric encryption, or secret key encryption. If M be a message to be encrypted, K the key to use, and $E_{key}(message)$ and $D_{key}(message)$ encryption and decryption respectively, for symmetric encryption we have [41]:

$$M \equiv D_K(E_K(M)) \quad (6.1.1)$$

Some encryption algorithms use two keys, where the knowledge of one is enough to determine the other, and these algorithms are also referred to as symmetric algorithms.

The security of a *symmetric* encryption algorithm is generally determined by the length of the chosen key (assuming no faults exist in the protocol), as the key length determines how long it will take to decrypt a cipher text, by trying every single key, until a match is found. This technique is also known as *brute force*.

Symmetric algorithms have the advantage that even short key lengths are secure from brute force attacks. A 56 bit key is no longer considered to be adequate (if security is a real concern), but an 128 bit cipher has not yet been guessed.

The most common symmetric encryption protocol has long been DES which was introduced by the National Security Agency (NSA) in 1976 and uses a key length of 56 bit. The use of this algorithm has not been recommended for some time, and in 2001 the AES algorithm was adopted by NIST as FIPS-197. Known as Rijndael until its adoption as a standard, this algorithm supports key sizes of 128, 192, and 256bit [36].

Because of the *shared key* design, two problems may arise when using symmetric encryption [39]:

- Key Management
- Key distribution

If a group of n need to communicate with each other, each member in the group would require $n - 1$ keys, or a total of $\frac{n \cdot (n-1)}{2}$ keys [36] (only half the total number because the keys come in pairs), which can quickly become unmanageable.

Key distribution becomes a problem because both communicating parties require the same key, which means that the key must be distributed through some secure channel.

6.1.1.1 Symmetric Encryption in Workflow Systems

In a workflow with n roles, symmetric encryption can provide confidentiality of the message exchange between

- $m = n$ roles and external entities
- $m \cap n, m < n$ roles, and any other entity.

If confidentiality only exclude external entities any role may access any message in a workflow, and if the confidentiality applies to a message exchange between two roles, no other role in the workflow can access this message.

In the first case, only one key is required, which is not unmanageable. The disadvantage to this approach is, that if that key is compromised so are all past and present messages in the workflow, as well as the security itself.

If $m = 1$ in the second case, it requires any role to possess $\frac{n \cdot (n-1)}{2}$ keys, because the role cannot know in advance with whom a message will be exchanged, and as the size m of grouped roles increases, so does the consequence of key compromise.

Hence, symmetric encryption can be used to ensure message confidentiality in a workflow, but it involves either complex key management and -distribution, or a very fragile security model.

6.1.2 Asymmetric Encryption

In asymmetric encryption algorithms, different keys are used to encrypt a message and decrypt cipher text. Given the definitions used in 6.1.1, and *encryption* key K_1 and *decryption* key K_2 , for an asymmetric encryption algorithm we have [41]:

$$M \equiv D_{K_2}(E_{K_1}(M)) \quad (6.1.2)$$

It follows that while K_{PUB} and K_{PRIV} belong together, knowledge of one cannot lead to knowledge of the other (otherwise it would be a symmetric algorithm given the definition in 6.1.1).

The most commonly used asymmetric encryption algorithm is RSA developed by Rivest, Shamir, and Adelman, and released in 1978, and is based on the problem of factoring large numbers [41].

Encryption using asymmetric algorithm is also called public-key encryption, because one key in the pair can be publicly known. Given an asymmetric algorithm that satisfies ($K_{PUB} = K_1$ and $K_{PRIV} = K_2$):

$$\begin{aligned} M &\equiv D_{K_{PRIV}}(E_{K_{PUB}}(M)) \\ M &\equiv D_{K_{PUB}}(E_{K_{PRIV}}(M)) \end{aligned}$$

Making one of the keys (K_{PUB}) publicly known, will allow:

- others to encrypt a message that can only be decrypted by K_{PRIV}
- the holder of K_{PRIV} to encrypt a message all other will know was encrypted by the holder of K_{PRIV}

This scheme reduces the problem of key management by an order of magnitude, as the total number of keys in a workflow with n roles is reduced to n keys¹, and the compromise of one key only compromises message confidentiality of messages sent to the owner of that key².

Using asymmetric encryption also mitigates the key distribution problem, because the key that is distributed can be publicly known.

6.1.3 Public-Key Authentication

Asymmetric encryption, or public-key encryption, can also be used to provide sender authentication and message integrity, by enabling the receiver of a message to verify that it was sent by the *holder* of a specific key, and that it hasn't been modified during transport. This is frequently used in communications protocols for client authentication (e.g. SSH) or message exchange (e.g. S/MIME), and are required properties in a secure workflow system.

Public-key authentication works by generating a cryptographic hash, $H(M)$, of the message to be sent, and this hash is then encrypted using the *private key of the sender*, which generates the

¹Using symmetric encryption with different keys for each direction of communication between two roles, would require $n \cdot (n - 1)$ keys.

²This resilience would also be achieved using $n \cdot (n - 1)$ keys using a symmetric algorithm[36]

cipher text [41]:

$$C = E_{K_{PRIV}}(H(M))$$

This cipher text is sent with the original message to the recipient, who performs verification by:

$$H(M) = D_{K_{PUB}}(C)$$

Because a hash value is generated on the message before it is sent, this also ensures message integrity as the this value is verified as part of the signature verification.

This allows a given role in a workflow to verify that:

- the correct key has been used
- the message has not been tampered with

6.1.4 Digital Certificates

Public-key encryption and its applications solves the key distribution problem, but it does not address the issue of whom the *owner* is. A recipient may know what key was used to sign or encrypt a message, but this doesn't provide any information about who the holder of the key is³.

In a workflow system, a role may be able to verify, that a specific key was used in the previous step, but since the role cannot verify the identity of the owner of the key, the role cannot be sure that the previous activity was carried out by the correct role.

Digital certificates is the common way of linking an identity and an asymmetric key pair. As the name implies, a digital certificate is a digital *certification* that the owner of this key is whom the owner purports to be [40]. Self-certification does not provide any confidence for the recipient so that an external party performs this action by signing the owner's public key.

If certificates exist that link all roles with their respective keys, and if all roles have access to all certificates, using digital certificates as identification would allow any role in a workflow to determine if an activity has been executed by the correct role.

6.1.4.1 Digital Signatures

Digital signatures⁴ and public-key authentication are closely related topics, and some works ([40], [39]) treat digital signatures as part of public-key authentication while others ([36], [41]) treat the topics separately.

A digital signature is intended to be a digital replacement for the traditional handwritten signature, and it should satisfy the following properties (among others):

1. Unique. No other combination of message and key may result in the signature $S' = S(M, K_{PRIV})$ ⁵
2. Unforgeable. Only the holder of K_{PRIV} , can create the signature S' .
3. Unalterable. Once the signature has been created it must not be possible to modify it
4. Irrefutable. When a signature has been created, the signer must not be able to refute the action.

³It is necessary to distinguish between *owner* and *holder*, as the key may be delegated to another entity

⁴This thesis only addresses *digital signatures* which is the application of public-key cryptography to the problem of digitally signing a message, and should not be confused with *electronic signatures* which address any means of creating an electronic signature.

⁵This also demonstrates one of the most common misconceptions about digital signatures. In the media a *digital signature* has become synonymous with a *digital certificate* issued by TDC. However, no two digital signatures can be identical, and it is the corresponding to the public key that was signed by TDC, that is used to *generate* a *digital signature* specific to what is being signed.

5. Not reusable. A previously signed message cannot be reused.
6. Fully Dependent. Changing any part of the message being signed, must result in the signature becoming invalid.
7. Verifiable. The receiver should be able to verify the validity of the signature

A digital signature scheme consists of an algorithm S to create a signature, and an algorithm V to verify the signature, and can be implemented with a public-key encryption algorithm combined with a collision-free hash algorithm, as described in section 6.1.3. This scheme doesn't satisfy the stated properties of a digital signature, because the identity of the owner of the key isn't known, which means the signature is reputable. This identity can be established if the owner is issued a certificate that attests the identity⁶.

The algorithms used to create digital signatures are not necessarily public-key encryption algorithms. The *Digital Signature Algorithm* (DSA) and the *Elliptic Curve Digital Signature Algorithm* (ECDSA), both standardized by NIST, are digital signature algorithms that are not used for encryption.

In a workflow, the properties that apply to a digital signature should also apply to the modifications performed in activity A and used as input in activity B . If these properties don't hold for data passed between activities, or if they cannot be proven (during activity execution, as well as later on), little trust can be placed in the transactions the workflow enable.

A simple example is the processing of an insurance claims form where the insurer will send the form to the insured, who fills out the form and returns it to the insurer. The insurer must be able to document that the insured filled out the form, and the insured must solemnly declare the truth of the information provided in the claims form, and to do so, the insurer will require the insured to send a signed physical copy.

Similarly, the transaction size (or trust) that can be supported by a given workflow system, is bounded by the trust that can be placed on the work performed in each activity (and the execution of the workflow process itself).

6.2 Legal Framework

The same properties that apply to a digital signature should also apply to the result of an activity performed by a role, hence

- the result of an activity should be signed by the role upon completion (6.2-1)

Organizations are to some extent free to accept risks that apply to internal processes, e.g. by accepting weak authentication controls in an internal workflow system. However, regulation ([2], [42]) is gradually increasing the requirements for companies to be able to document effective internal control procedures (e.g. that strong authentication is being used), and when transactions cross organizational borders, most organizations require a more formal framework for managing risks inherent in transactions. The traditional approach is based on paper letters and physical signatures, and it is desirable to extend the advantages of this system, to a digital workflow system.

The rest of this section provides an overview of the possibilities, that current legislation allows for use of digital signatures as a replacement for physical signatures.

⁶As a side note, this is the difference between public-keys used by SSH, and e.g. X.509 certificates

6.2.1 Current Legislation and Market Adoption

In Denmark, two documents address the legal force of *electronic* signatures, and these documents also address *digital* signatures:

Law on Electronic Signatures (L417) This law [3] was the first Danish law that addressed electronic signatures, and despite its name, the law *only* applies to signatures generated with *qualified* digital certificates (§2). The law implements the directive 1999/93/EF from the European Parliament, and besides regulating operation of certification authorities (CA), the law also acknowledges (qualified) certificates issued from foreign CA's (§23). The law was passed in 2000, and does not apply to OCES certificates, as they are not qualified.

Report no. 1456, The Force of Law of Electronic Signatures (B1456) This report was issued by the Danish Justice Department in 2005 [4], and addresses the legal force of several different electronic signature schemes including digital signatures. The report was the first clarification of the legal force of signatures created using an OCES certificate, and establishes the legal force as the same as a physical signature, but also precludes their use in some cases (e.g. trading real estate).

OCES certificates are not qualified⁷ certificates but are issued by a certification authority that must adhere to the requirements stated in the certification policies (CP) controlled by the National IT and Telecom Agency⁸.

To summarize, using digital signatures with legal force is possible with:

- OCES certificates
- Qualified certificates

As law 417 recognizes certificates issued by a foreign CA, using qualified certificates has a number of advantages in terms of *crossing borders*, because foreign certificates can readily be accepted, and the directive on which it is based has been adopted by a number of European countries.

However, currently the only (Danish) certificate authority issuing certificates is TDC Certificeringscenter, and TDC stopped issuing qualified certificates in 2005, meaning that only OCES certificates are readily available in Denmark.

If a workflow system is deployed in a company with subsidiaries in other countries, the choice of "certificate" may present a problem, as it can determine in which countries the process execution (transaction) is binding. If the transactions are only relevant in the country of the mother company, the chosen certificate only needs to be recognized in its residing country, which reduces the problem to choosing "certificates" in the subsidiaries. Otherwise, a common denominator must be found¹⁰.

Given the market adoption of OCES certificates, the best approach to facilitating transactions in a workflow system that have legal force appears to be to use OCES certificates in Denmark, and in subsidiaries to select certificates issued by a CA whose disclosed certificate policy (CP) and certificate practice statement (CPS) are along the lines of L417.

In any case, systems enabling transactions signed with digital signatures with legal force is a new area, given that there have been no court rulings directly addressing the topic of the legal force of digital signatures [44]. There have, however, been several court cases addressing the

⁷A qualified certificate refers to a certificate, that is only issued upon personal appearance.

⁸In March 2003 the Danish government afforded TDC a \$7 mill. tender to establish a national PKI infrastructure dubbed OCES. The intention of this tender is that all Danish citizens should be issued a digital certificate signed by a CA that is in compliance with a relevant OCES CP [43]. The tender does not mandate, that TDC be the only CA issuing OCES certificates, and indeed EuroTrust, and not TDC, was the first Danish CA to be approved for issuing certificates according to an OCES CP⁹. OCES certificates are issued according to three OCES CP's, that are controlled by the National IT and Telecom Agency through "Signatursekretariatet".

¹⁰assuming that both countries have passed the necessary laws, to enable digital signatures.

legal force of transactions, in which the trust in the IT-systems has been established through a systems analysis, and any disputes involving digital signatures will likely also be resolved using this approach [45].

6.3 Current PKI Models

Public-key encryption solves the problem of maintaining key secrecy in key distribution, but it does not solve the problem of actually distributing or managing the (n) keys, and to use public-key encryption for digital signatures introduces the requirement, that (public) keys must be signed by a certification authority. The solution to the key distribution problem is determined by the key certification system that is employed, because the certifying authority also is responsible for distributing a certificate to the requester.

Public-keys are commonly distributed through a *public-key infrastructure*. The two most common public-key infrastructures, are PGP and X.509¹¹. In PGP there is no central certification authority and instead all principles can sign keys. Trust¹² in a specific key is then resolved by looking at the signatures attached to this key. X.509 works the opposite way, by basing all decisions of trust relationship on a single certification authority.

PGP is widely used to secure e-mail communications, but all [47] technology specific legislative work concerning digital signatures is currently focused on X.509 certificates, hence the system should ensure that

- the result of an activity must be signed with an X.509 certificate (6.3-2)

6.3.1 Trusting a Certification Authority

Using X.509 certificates, the decision to trust the identity purported by the holder of a key is reduced to deciding if the signing certification authority can be trusted to correctly identify the role. As the recipient has no way of directly determining this, existing CA's establish their own trustworthiness, by

- promising monetary compensation for any losses related to trusting an erroneously identified role
- submitting to external audit by an independent party (whose professional insurance indirectly covers any claims)

All major national and international CA's (VeriSign, GlobalSign, TDC Certificeringscenter) do both. In Denmark CA's issuing OCES certificates are required to comply the OCES certificate policies, and this compliance must be audited by an external independent auditor.

The binary trust model between the signer and the signed X.509 certificate means, that *all* certificates issued by a trusted signer will be trusted. As many operating systems and applications come "preconfigured" with a large number of certification authorities, it means that most base system installations (e.g. Windows, Mac, Linux, or Java) will trust a large number of certificates without any "opt-in" on the part of the user.

In a workflow using X.509 for identification of roles, the assignee of an activity is determined both by a role's certificate, and simply trusting the certificate will not have any adverse affect on this. However, from a principle of least privileges, a secure workflow system should require

- specification of all trusted certificate signing authorities (6.3-3)

¹¹ X.509 and related standards, e.g. certificate revocation lists (CRL), are developed by the PKIX Working Group [46] under the Internet Engineering Task Force (IETF).

¹² The trust establishment that is referred to, is the degree to which it is possible to trust that a given key represents a given identity

6.3.1.1 Hierarchical Certificate Authorities

The problem of resolving trust in an X.509 is further exacerbated, if *deep* certificate hierarchies are used. In the standard model one CA signs all certificates which creates a two level key hierarchy, that is a completely flat model. However, additional levels could be introduced to reduce the width of the hierarchy and remove work from the single signing CA¹³.

If all adjacent levels in a hierarchy cross certify each other, tree traversal will enable trust resolution [40] albeit few existing systems offer this functionality.

Supporting trust resolution in deep certificate hierarchies is not a significant requirement of a workflow system that uses certificates. Given the intended application domain it is unlikely that hierarchies will be required, and the OCES PKI is currently limited to a two-level hierarchy, as an OCES CA must be the root in its hierarchy [43].

6.4 Summary

Public-key encryption and the related digital signatures can guarantee a number of desirable properties about activity execution in a workflow, and current legislation makes it possible to use digital signatures that have the legal force of a handwritten signature. The Danish legislation also opens the possibility, that certificates signed by foreign CA's can be used with the same legal force in Denmark, which is import in the case of organizations with subsidiaries in other countries. In Denmark OCES certificates provides a common PKI based on X.509 certificates.

¹³This is a common *tactic* employed by commercial CA's, as it provides a cheap way of being adopted in root certificate stores shipped with e.g. operating systems. A certification authority creates a new CA solely for the purpose of signing other CA's, and because this *super* root only needs to signing a handful of certificates, *very* stringent operational procedures can be put in place around this super root, making it fast and cheap to have assessed by and independent auditor. Being adopted in e.g. Microsoft's root certificate store is possible if e.g. the audit resulted in the *certification authority* being issued a seal according to an official attestation standard (e.g. AICPA WebTrust, or SAS 70), and typically if a CA is in compliance with on of the aforementioned standards, all CA's signed under by the compliant CA will be adopted as well.

7

Requirements Capture

In this chapter requirements for the system are analyzed. Section 7.1 analyses use case driven system requirements, and section 7.2 and 7.3 expand these to include external requirements.

Chapter 2 referred to the file that a user exchanges with the system, as a *container* that contains the document the user modifies, and these definitions will be used from this point forward.

7.1 Use Case Analysis

Figure 2.0.1 in chapter 2 showed the two core use cases for the system, from the point of view of a user. This section expands on these, and adds use cases for other roles of the system. Use case diagrams corresponding to the identified use cases are provided in appendix C.

This use case analysis looks at the interaction between an end user and the system, and does not address the interaction between a system administrator and the system. However, requirements for system management are specified in section 7.2.

The user interaction can be summarized as getting a container, and submitting a container, and these cases can be decomposed into user-system interactions.

The following table shows the step that must be completed for a user to get a container from the system:

Get Container	
Actor Action	System Response
1. Login to the system	2. Display list of waiting containers
3. Select container	4. Send container to user
5. Verify document signature	
5. View information about container	
6. Read document in container	

The corresponding use case diagram is figure C.0.1, appendix C.

The following table shows the step that must be completed for a user to submit a container from the system:

Submit Container	
Actor Action	System Response
1. Process a document in a container	
2. Sign document	
3. Login to the system	
4. Upload modified container	5. Acknowledge submission

The corresponding use case diagram is figure C.0.2, appendix C.

Processing a document in a container means, that a user is required to perform one of the following three actions on the document:

- Modify a document in a container (7.1-1)
- Review a document in a container (7.1-2)
- Finalize a document in a container (7.1-3)

Creating and modifying a document is taken to be the same action as it is assumed, that if a user can modify a document, the user can also replace all content in this document, effectively making it a new document.

Summarizing the user-system interactions, the user must be able to perform the following actions:

- Login to the system (7.1-4)
- Logout of the system (7.1-5)
- Sign document in container (7.1-6)
- Verify signature (7.1-7)
- Download a waiting container (7.1-8)
- Upload modified container (7.1-9)
- List waiting containers (7.1-10)
- Process a document in a container (see actions above)
- View information about a container (7.1-11)
- Read a document in a container (7.1-12)

7.2 Workflow

This section analysis the necessary functionality required to support workflows that meet the objectives of section 4.4.

7.2.1 Workflow Activities

In section 7.1 the possible activities were identified as *modify*, *review*, and *finalize*, and these are the activities that should be supported.

Of these activities only *modify* involves changing the actual document, whereas the other two only changes *information* about the document; *review* will attach a status to the document, and *finalize* will end the workflow. An activity should capture all changes and additions including:

- Document modifications (7.2-1)
- Review categorization (7.2-2)
- Document status (7.2-3)

In chapter 6 it was shown, that it is necessary that a role signs the work completed in an activity. A signature represents an act of commitment, however what is committed to must also be expressed, and this also applies in a workflow system. The aforementioned properties capture the commitment of the signing role in an activity.

Execution of an activity will also generate information about the execution, or the user may wish to supply data about execution, hence an activity should also capture the following as part of the activity execution:

- Automatically generated, structured meta data about the activity (7.2-4)
- Unstructured meta data about the activity supplied by the user (e.g. comments or keywords) (7.2-5)

Finally,

- all properties must be included in the signature scope (7.2-6)

If all properties that are used as part of the workflow execution are not signed, the trust that can be assigned to each property will vary, and defies the intended level of trust.

7.2.2 Process Support

Process support determines how complex workflows are allowed to be, as well as the ease with which complex workflows can be expressed. Support for certain complex transition patterns may make expressing certain workflows much easier, and more importantly requiring fewer steps to complete¹ [20].

7.2.2.1 Transition Patterns

The objective of the system is to support the simple transition patterns, and the system should support the following transition patterns listed in decreasing order of importance:

	Objective	Pattern	
1.	4.4-3	Sequence	(7.2-7)
2.		Parallel Split (AND-split)	(7.2-8)
3.		Synchronization (AND-join)	(7.2-9)
4.		Exclusive Choice (XOR-split)	(7.2-10)
5.		Simple Merge (OR-join)	(7.2-11)
6.	4.4-5	Multi-choice transition pattern	(7.2-12)
7.	4.4-6	Synchronizing merge transition pattern	(7.2-13)

Patterns 2+3 and 4+5 are mutually inclusive, and patterns 6+7 renders 4+5 superfluous.

¹Simple transition patterns can express most classes of workflows, but the expression may be very verbose.

7.2.3 Signature Scope and Ordering

In a workflow multiple roles will sign different versions of documents within a container², making it necessary to define:

- The scope of a signature by a role
- The ordering of signatures if parallel processes occur

To support the notion of a workflow, a role must always commit to the current state (before review or modification), thus acknowledging the roles assignment. The scope of a signature must:

- Include the signature of the previous activity (7.2-14)
- Include the work of the current activity (7.2-15)

By definition, to include an activity (A_1) in the scope of a signature on another (A_2), that activity must complete at time $t_1 < t_2$. Consequently, the scope of a signature can never include a concurrent activity, because only synchronized parallel activities are supported, which means that by definition two activities always enable the next activity at $t_1 = t_2$.

7.2.4 Error handling

Error handling in a workflow system determines the system's response to an unexpected (application logic) state, and by extension how gracefully errors are handled. An error state is defined as the current application state at time t_2 , at which time it is discovered that data accepted at time $t_1, t_1 < t_2$ is invalid for the current state.

Not all invalid data creates an error; when a user is not logged in, the system should just remain in this state, until valid credentials have been presented, which also implies, that an error state can only occur when a user has been authenticated. If a user submits a container with an invalid signature this creates an error state, because this state isn't possible, as the system should prevent creating invalid signatures.

The systems response to an error state must always be:

- Stop workflow processing (7.2-16)
- Log the error (7.2-17)
- Notify the system administrator (7.2-18)

An error state is either an error or an intentional act, and in either case it should require an active decision to dismiss the error.

As a significant amount of time and resources may be invested in the (partial) product in a workflow system, when an error state occurs, it is important to prevent loss of data, and limit the cost of re-performance, hence:

- The result of a completed activity must never be lost due to an error state (7.2-19)
- An error state must only require re-performance of the affected activity (7.2-20)

As all workflow products should be versioned (section 4.4.3), these requirements are implicit properties of the data model³.

²As defined in the workflow objectives, a workflow only includes one container.

³The implementation will obviously affect these requirements

7.2.4.1 Signature Error States

The following combinations of certificates and signature validity are possible:

	Valid Sig- nature	Invalid Signature
Correct certificate	Ok	Error
Incorrect certificate	Error	Error

A correct certificate is the certificate identifying the role an activity is assigned to.

An incorrect or invalid signature generally always cause an error state. Choosing an incorrect certificate or producing an invalid signature is easily detected by the signing application, and processing should not be allowed to proceed past this detection point, hence, either occurrence is itself an error that must be investigated as they likely represent application errors or intentional actions.

7.2.5 Workflow Specification

A workflow specification controls a workflow process and should explicitly specify all aspects of the process, that must be controlled. How a system allows the workflow specification to be created and reused between workflow executions determines how the workflow system can be used.

The graphical workflow description languages described in chapter 4 are not necessarily the best approach to specifying a workflow process that can be translated into an application state, as neither Petri Nets nor UML activity diagrams makes it easy to specify properties such as access control.

Requirements related to workflow specification can be divided into requirements related to the expressability of the specification, and how a specification is used by the system.

Limitations

A workflow specification tool is outside the scope of this project given the size and complexity of such a tool (see [48] or [49] for examples of graphical workflow specification tools), and requirements for such a tool will not be addressed.

Terminology

A workflow process is *executed* according to a *workflow specification*. This specification is *created* once for each workflow process, and once a specification is created, a new workflow can be *instantiated* from the specification.

7.2.5.1 Creating a Specification

As described in [20] accurately capturing and specifying a workflow process is a complex task that requires careful analysis of the involved work, and is generally delegated to specialized personnel. A process that is carried out infrequently or in very different ways is not a suitable candidate for (this type of) workflow automation, rather the process execution path of the process should be predictable and well defined, and the process description should not change very often.

Syntax

A workflow specification must be expressed in terms of a syntax, and this must be structured so as to allow automated processing, as well as conversion into another workflow specification language (e.g. a graphical notation language, or a different workflow specification language such

as BPEL).

The specification language itself must be able to:

- Express all activities stated in 7.2.1 (7.2-21)
- Express all transition patterns stated in 7.2.2 (7.2-22)
- Contain all relevant production- and control data (7.2-23)

As requirements for a workflow system will eventually evolve, the specification language should also:

- be extensible to allow new features to be added at a later stage (7.2-24)

7.2.5.2 Using a Specification

A workflow is often applied to regularly occurring processes, hence

- The system must allow reuse of a workflow specification (7.2-25)
- Given proper authorization, a user should be able to instantiate a workflow from an existing specification (7.2-26)

Before a workflow can be instantiated it must be specified, and once a workflow is specified, this specification can be used to instantiate any number of identical workflows.

7.2.6 Workflow Administration

This section details the basic administrative features required by a workflow system implementing the specified system requirements, and is not a full analysis of the requirements for system administration.

A workflow system can be designed to allow or disallow modifications to the workflow process while the workflow is executing. Allowing modifications at run-time makes it easier to resolve halted processes, but also makes it possible to circumvent the checks, that the system is designed to enforce. Because the product of the workflow system should be self documenting and all signatures should remain valid, disallowing changes also simplifies the implementation, hence:

- Changing the workflow specification of a workflow instance must not be possible (7.2-27)

Exceptions to process execution may arise, that requires immediate intervention action, which means the system should allow an authorized role to:

- View *all* containers (7.2-28)
- Delete an *active* container (7.2-29)

It is important to note, that while an administrator should be able to delete all active containers, an administrator should *not* be able to delete containers that are the result of a completed workflow.

Finally, it must be possible to manage all roles (users) that are part of a workflow, hence an administrator must be able to:

- View all users (7.2-30)
- Create a user (7.2-31)
- Modify a user (7.2-32)
- Delete a user (7.2-33)

7.3 Security

This section analysis the necessary functionality required to support workflows that meet the objectives of section 5.4.

The trust model of the system is defined, and the security requirements are analyzed in terms of the five common security metrics [40].

7.3.1 Trust Model

The basis of all security properties ensured by the system is the underlying trust model. This system will be based on:

- A binary trust model in which any user that is authenticated is trusted. (7.3-1)

Currently most systems employ a binary trust model, in which a resource is either trusted or not, however much research is focused on:

- Incremental trust models in which a resource is gradually trusted [50]
- Decision making systems in which decisions of trust are evaluated at run-time (e.g. KeyNote [51] and PolicyMaker [52])

Both approaches offer advantages that cannot be achieved using a binary trust model. In the OpenBSD operating system IPsec is implemented using the KeyNote decision making system, thus enabling variable network trust without any user interaction of system configuration [53].

However, the coordination of user activity is by definition structured and predictable, and the IT-system will operate in a controlled and monitored logical environment, as can be expected of a modern IT-infrastructure. Finally, availability is not a primary requirement (see 7.3.4).

This means that neither the system nor its operating environment have many of the characteristics of a setup where an incremental trust model would be desirable, and a binary trust model seems to offer a better compromise between system security and complexity of implementation.

7.3.2 User Identification (Authentication and Signature)

Expressed informally, anyone accessing the workflow system or any related data should be authenticated. Authentication must be done using a signature scheme (see chapter 6), and is the process of presenting a valid signature.

The detailed requirements for authentication and signature is summarized by the following

- All participating roles must be registered with the system (7.3-2)
- A role is identified by the DN of that role's certificate (7.3-3)
- To login a role must provide a signature using a valid signature scheme (7.3-4)
- A role must re-authenticate to gain access to control and production data that is newer than the current authentication (7.3-5)
- An activity by a role must be digitally signed using a valid signature scheme (7.3-6)
- When an activity has been committed by signature, neither activity nor signature may be revoked (7.3-7)
- Any role must be able to authenticate the activities of all previous roles (7.3-8)

The stated properties assume correct authorization.

Property 7.3.2-3 simply expresses that once a role has been granted access to data, it is to be as-

sumed that that role will always have access to that data (e.g. through a copy), and is a necessary restriction to ensure valid authentication to current data.

Because roles are required to include the result of the previous activity as part of their own signature, it is important that a user is able to verify the state of the workflow execution which property 7.3.2-7 ensures.

7.3.2.1 Signature Scheme

A primary requirement of the workflow system is that activities have legal significance, hence the *chosen* signature scheme must be encompassed by [3] and by extension [4].

This imposes the following requirements on the system implementation:

- It must be possible to use X.509 (7.3-9)

Because of the difficulty in graphically reproducing content the same way across different systems, this is cannot be required of the supported system, hence:

- The system need not support what-you-see-is-what-you-sign (7.3-10)

7.3.3 Access Control (Authorization)

Access control can be divided into controls that can be enforced only when the container is stored in a physical machine to which a role does not have access, and those that can be enforced even if the role has physical access to the machine.

It is not possible to prevent⁴ a role from changing a container when the role has control over the container, even if the access controls specify that the role is not allowed to do so. However, it is possible to prevent a role from reading a document in a container by encrypting it.

Requirements for authorization is summarized by the following

- Any role must be able to confirm the authorization of all previous roles (7.3-11)
- A role must not be able to effect any loss of data other than the data that role has created during the current activity (7.3-12)
- A role must not be able to effect loss of data that role has created during a previous activity (7.3-13)
- A role must be authorized to perform an activity (signature) (7.3-14)

7.3.4 System Availability

It stands to reason, that availability requirements for a given system depend solely on the purpose of that system, and the requirements for system availability should be evaluated in terms of the effect, that an unavailable system has on the organization employing this system.

Requirements for availability is summarized by the following

- System unavailability has negligible impact on the surrounding organization (7.3-15)
- Complementary security technologies can thwart attacks by unauthenticated users (7.3-16)
- Implementation should allow transparent redundancy or fail-over (7.3-17)
- Implementation may allow limited operation without infrastructure (7.3-18)

⁴The problem of enforcing access control on digital content in the (complete) control of a user, can be likened to the problem of enforcing DRM on other digital content such as music and films, which so far has proven futile. The system design will assume that a user is able to perform *any* action on digital content in the user's control, but the user will not be able to read encrypted content, *the user is not nor has been allowed to read.*

7.3.4.1 Impact of Unavailability

A document workflow system in which each role is a person, and where the work implied in each activity takes much longer to complete than the role-system interaction of that activity itself, does not carry any special requirements for system availability (assuming the domain description of 2).

If each activity involve work that takes one or more people an average of several hours to complete, a system unavailability of a few hours will be insignificant compared to unexpected delays that execution of a given activity may involve. One activity in a workflow may e.g. involve registration of results of a lab experiment, however this experiment may fail thus delaying the *entire* workflow for the duration of this experiment.

It may further be assumed, that the system will be operated in the context of a modern IT-infrastructure, employing operational procedures and monitoring intended to register system errors and alert system operators.

This means, that system resilience to attacks aimed at system resources and system availability is not a *primary* requirement as

- system availability doesn't have any noticeable effect on the surrounding organization
- compensating controls in the general IT-infrastructure will detect this

7.3.5 Document Confidentiality

Document confidentiality controls who can read the versions of a document a workflow execution accumulates. Most companies use e-mail for communicating confidential information, but very few actually use encrypted e-mail, hence using that as a baseline for requirements for confidentiality, would indicate that requirements are limited.

A more realistic baseline requirement is:

- An unauthenticated or unauthorized role `must` not be able to access any workflow data (7.3-19)

Document confidentiality can also be extended to implement reader access authorization, in which case the following apply

- A role may only access production data for explicitly defined activities (7.3-20)
- A role may only access control data for explicitly defined activities (7.3-21)
- A role may only know the identity of explicitly defined roles (7.3-22)

If document confidentiality is ensured by encrypting each unit of content, regardless of who gains access to a container, they would still require the proper key(s) to access the content.

7.3.6 System Auditing

The main requirement for system auditing is:

- Workflow execution must be self documenting (7.3-23)

System auditing facilitates debugging, and ensures a complete audit- and transaction log⁵ which is a requirement.

The following properties apply to system auditing:

⁵Maintaining complete audit- and transaction log is required by law for systems that process data that affects a company's financial statement.

- Workflow execution should be audited (7.3-24)
- All signatures should be present in a document (7.3-25)
- The system should audit relevant system actions to facilitate fault finding (7.3-26)
- The system should use an industry standard log facility (7.3-27)
- (Optional) the server should sign all log lines (7.3-28)

Using a standard log facility is important because it allows the system logs to be collected, correlated with other log material, and analyzed centrally.

7.3.7 Version Control

The system must version control all role submissions to make it faster to recover from an error, and to make it possible to verify signatures from previous activities. The version control must ensure:

- An error does not set the process further back, than the last completed activity (7.3-29)
- A complete copy of all versions is always stored in a way accessible only to a system administrator (7.3-30)

The *current* container in a workflow is defined as the last container that was successfully submitted, before a new one is submitted.

7.4 System Architecture

This section analyzes requirements of the system architecture of a system that implements the intended secure workflow system. Existing requirements for workflow and security indirectly also imply requirements in terms of system architecture (e.g. OCES certificates should be supported, hence the system must support X.509 digital certificates), but it is also necessary to define the target platform on which it should be possible to use the system, as well as the scope of document types with which the system can be used.

7.4.1 Document Support

Whether a document workflow system should be part of the actual content modification process is a basic choice of functionality. A financial system is an example of a system that includes a facility for creating and approving an invoice according to a specified process, whereas communicating a Word document by e-mail between author and reviewer is an example of separation between the two domains.

The system is intended as a generic document workflow system, and should support different application domains, hence

- Content modification should not to be integrated in the system (7.4-1)
- The system should support all document formats (7.4-2)
 - A document must not depend on external references
 - A document must be storable as a single, machine independent file⁶
- Document storage must be formally specified (7.4-3)
- Document storage specification must be system independent⁷ (7.4-4)

Requirement 7.4-2 enables the workflow system to support any number and types of files that can be combined in an archive format such as TAR, ZIP, or RAR.

⁶Byte order or similar properties must not limit the use of a file to a specific machine hardware architecture

⁷For practical reasons the specification will (obviously) not be character set independent

The product of a workflow execution must be accessible for the duration of time in which the product is used, including decisions that are directly based on the (content) of the workflow product. The run-time may extend beyond the lifetime of the system, in which case it should be possible to recreate the result of a workflow. Depending on the application domain, this may also become a legal requirement (the new Enterprise Technical Reference Model recently adopted by the state of Massachusetts [54], and the Valoris report commissioned by the European Commission [55][56] being recent examples).

7.4.2 Platform Support

The system can be divided into a part used by a user (the client)⁸, and a part used by an administrator (the server), both of which have different requirements for platform support.

It is assumed that the server is run (is used) in a typical server infrastructure, which means the server should:

- Be possible to run on a typical server operating system (7.4-5)
- Not require any special hardware (7.4-6)
- Require limited administration (7.4-7)
- Be easy to integrate in the existing infrastructure (7.4-8)

The client must be usable on the operating systems commonly used by the intended users, which includes support for:

- Linux (7.4-9)
- Apple Mac OS X (7.4-10)
- Microsoft Windows (7.4-11)

It will be assumed that no malicious code is executed on either the client or the server, hence the implications of malicious code on the system are not treated. The architecture of current operating systems provide limited protection from malicious code, if the code is executed by the user or with system privileges by exploiting a flaw in part of the system.

Finally, it is assumed that the server can always be trusted from the point of view of a client.

7.4.2.1 Resource Constrained Devices

A resource constrained device is defined as a device that has significantly lower computational power than typical desktop computer (in 2005), which includes e.g. cell phones, and PDAs.

The factors determine platform support is relevant

1. Does the users require support on such a device
2. Does the resource constraint incur an unacceptable additional processing time
3. Are all program libraries used in development available on the target platform

Given the increasing popularity of PDAs and similar devices, it should be expected that users will request support from these platforms (1), and given adequate demand (3) is simply solved by implementing the missing functionality (or rewriting the program to work without).

(2) is interdependent with the implementation, as the implementation can be optimized for a resource constrained device. However, requirements 7.3-18 - 7.3-21 implies encryption of all

⁸This definition only states the intended user, and does not assume anything about where the actual program execution takes place. As such a client can be a program that is executed locally, or it can be a terminal emulator.

content, and currently PDAs (not to mention less more constrained devices) do not have the computational power to encrypt and decrypt *several megabytes* of data fast enough to be useful.

A Palm Pilot (Tungsten T) equipped with a Texas Instruments ARM 925 processor (432 MIPS according to the published specifications) can encrypt a 389 kb PDF in approx. 53 seconds using a Blowfish-128 symmetric encryption algorithm. The performance scales linearly, and encrypting 2533 kb takes 6 minutes 4 seconds.

Dumb terminals (VNC, RDP, etc.) should not be supported, as the current terminal protocols only offer limited access to local certificates. Using RDP it is possible to access a PKCS11 device⁹, but this will only work on Windows clients, and PKCS11 devices are not commonly used.

⁹An interface specification for accessing cryptographic hardware devices, such as a Javacard smart-card.

8

Container Specification

The overall design goal of the system that the final product of the workflow should be self documenting (section 1.2) mandates that information about the workflow execution (control data) should be transmitted together with the actual production data, and that the two must be inseparable.

The previous chapter referred to the information being transmitted as a *container*, in which the actual document was stored. A container must store all versions of relevant control and production data, and this chapter describes a specification of a data model, that supports a self documenting workflow execution that satisfies the requirements stated in the previous chapter.

Terminology

This chapter describes a *specification* of a format for a container. From this *specification* a *container instance* can be *instantiated*, which can be used in a workflow execution.

8.1 Container Format

A container instance must be stored in a well defined format (the file format), that is open¹ and machine independent.

- the content of a container will be stored as text (UTF-8) based XML ² (8.1-1)

The fact that XML is text based can also be a disadvantage to the format, as it makes it less space efficient than e.g. a binary file format, and e.g. ASN.1 (Abstract Syntax Notation) could have provided the structure of XML combined with the efficiency of a binary file format.

The penalty in terms of space for using XML instead of a binary format is approximately a 30% increase in file size³, but the ease of working with XML compared to e.g. ASN.1 more than offsets this.

Using XML also incurs a processing overhead, as parsing an XML file is slower⁴ than just reading bytes from a stream, but because the content is structured it is possible to use the XML parser to (1) validate that the content is well formed and (2) validate the content to the extent allowed by the chosen XML specification language.

¹The file format specification itself should also be open, to satisfy requirement 7.4-3

²Binary variants of XML have also been standardized, but in the following an XML file is assumed to be text based.

³base64 encoding a binary file produce a (UTF-8) text file that is approximately 30 % larger, than the original binary file.

⁴Using Java and Xerces, parsing a 24 kb byte array stored in memory into a DOM takes approx. 200ms

8.1.1 Versioning Model

A container instance must contain all previous document versions, and the container is text based - and combination that may result in impractical container file sizes. Common version control systems (e.g. SVN or CVS) limit this problem by only storing the difference between the current and the previous document, but most version control systems cannot version binary files (which is the most probable content for this workflow system), and supporting incremental binary versioning requires a more complicated binary patching algorithm.

If the *X-flow* system is assumed to be used primarily with commonly used office⁵ file formats, the average file size is 400kb⁶, the ratio of modifications to reviews is 1 : 3, and a digital signature is 4kb, the storage requirements specified in terms of the number n of steps in the workflow is:

$$\begin{aligned} filesize &= \frac{1}{3} \cdot n \cdot (400kb + (4kb + 4kb)) + \frac{2}{3} \cdot n \cdot (4kb + 4kb) \\ &= \frac{1}{3} \cdot n \cdot (400kb) + n \cdot 8kb \\ &\approx n \cdot 141.3kb \end{aligned}$$

In a workflow in which 20 activities are executed, that would result in a container file that is \approx 3Mb, and parsing such a file is not a problem.

8.2 Specification Format

If an XML format is specified using a formal specification language⁷ (an XML schema) that is supported by an XML parser, it can be used to validate the properties of the XML file that can be expressed using the chosen language.

- The syntax of the container file format will be specified using the XML Schema Language (8.2-1)

There are currently 4 XML schema languages supported by commonly available XML parsers:

- Document Type Definition (DTD)
- XML Schema Language
- Relax NG
- Schematron

DTD is the oldest of the 4, and is borrowed from SGML. It lacks many of the features of the others, and is not relevant. Schematron offers great flexibility in testing documents, because any test that can be expressed as a binary XPath query can be tested, however the language is not modular in its design making it difficult to work with large schemas [57]. Relax NG adds modularity, but does not have the expressive tests offered by Schematron, and also does not allow default attribute values, which is desirable to prevent undefined values from appearing. XML Schema Language is W3C standard, and while it does not offer the extensive tests⁸ of Schematron, it provides default attribute values, and is the best compromise between the feature set of the three.

The specification (XML schema) of a container (XML instance) is the data model of the system, and in the rest of this chapter, *data model* and *container schema* will be used interchangeably.

⁵Not referring to the Microsoft Office product, but rather files that are used by a typical office worker, which precludes e.g. large image files

⁶This number was found as an average of more than 3.000 files produced from 2000-2005 (courtesy of PwC)

⁷An XML format could also simply be specified using an informal textual specification.

⁸XML Schema Language also offers XPath based tests, but their working scope is limited to subtree of the parent node of a given element definition.

8.3 Data Model Design

Designing a data model represents a compromise between which system features that are controlled by the data model specification, and which features are (only) controlled by the system design.

The container schema is designed to explicitly define (thus control) the operation of required features of the system (e.g. when a document is required to be signed). This allows using an XML parser as model checker to verify that a container represents a correctly executed workflow⁹.

8.3.1 Workflow Support

Workflow support can be enabled by using an existing workflow engine and its syntax, or the system can use its own workflow engine, with a suitable syntax:

- The *X-flow* system will use its own workflow engine and specification syntax (8.2-1)

Using an existing workflow engine has the obvious advantage that it is already implemented (the *engine*), which means less code to develop and maintain. However, using an external engine means relatively more complex integration compared to using an engine that is designed to be part of the system, and if the syntax does not encompass all elements of the application domain, a new syntax, and a translation mechanism, must be developed to bridge between the syntax of the chosen workflow engine, and the syntax requirements of the *X-flow* system.

8.3.2 Verification Protocol

The data model must support a verification protocol that can be used to verify the process execution at any given step in the workflow. Verification should include all formal requirements of the workflow, including:

- Execution order
- Role authentication and authorization

The verification protocol should also limit the amount of work required to perform the verification (preferably, execution time should always be $O(1)$ bound).

8.3.3 XML Signature

Support for digital signatures is done using XML Signature. XML Signature is a standard adopted by W3C and is used to express a digital signature as XML. Using XML Signature it is possible to express a signature in three ways:

- Enveloping. In which the signed content is included *within* the the XML Signature element.
- Enveloped. In which the XML Signature element is a child of the signed element
- Detached. In which neither the XML Signature element nor the signed element is a descendant of each other. The signed content may all be an external reference.

An XML Signature references the content (XML or not) to be signed by an URI, and several references can be included as part of a signature.

The container schema only uses enveloped and detached signatures. When detached signatures are used, it is only because the referenced element is located in a different part of the tree, and external (to the XML instance document) content is never signed¹⁰.

⁹With regards to the features whose behavior can be expressed in terms of the XML Schema Language.

¹⁰This also means that a signed URI is always on the form `#<element xml:id>`

XML Signature requires quite a lot of code to create, parse, and load XML structures, and because it is a newer standard¹¹ most programming languages (e.g. Java) don't support the XML Signature specification in their native libraries. On the other hand signing and encryption operations using algorithms such as RSA and DSA are readily supported by most common programming languages, and implementing signature support using such operations would result in a more widely supported system.

However, the Apache Foundation provides free libraries for C++ and Java supporting XML Signature, and native support is being worked upon, e.g. also for Java, and the standard provides the core security of SOAP hence support in other languages can be expected.

8.4 XML Schema Design

This section contains a detailed description of the design of the container schema. The full schema specification (`container.xsd`) is included on the CD-ROM in the directory:

```
$CDROM/source-code/resources/schema/.
```

8.4.1 General Structure

The following listing shows the generalized XML structure of a container instance (the listing is *not* schema valid).

```
<container>
  <info/>
  <workflow/>
  <documents/>
  <transactionlog/>
</container>
```

<container> Is always the root element of a container instance. This element must always include the definitions stated in section 8.4.8

<info> Specifies the server endpoint that clients should use in a workflow. This element (and children) includes network information needed by a client to contact a server, as well as identification information about the entity that assigned this `<info>` element, which is not necessarily the server endpoint. It is not a requirement that the same server endpoint be used for the duration of a workflow execution, and this element may be updated as needed. The `xml:id` of this element is *always* `info`, which is enforced using a `final` value of the `xml:id` attribute. The element must always be signed with an XML signature that is an immediate child of the `<info>` element, and which [the signature] *only* includes a reference to the `<info>` element id.

<workflow> Contains a complete specification of the workflow process in which this container is used, including a description of the task (modify, review, and finalize) in each activity, and role authorization. Once a workflow is started, it is not allowed, to change this element and this will invalidate the entire workflow execution.

<documents> Contains all document versions that have been created during the workflow execution. A document is non-static which means that storing outside the hierarchy of the workflow specification is preferred. Each new document version, or each review that is added to the container, is added as a `<document>` element as a direct descendant of this element.

<transactionlog> Is an audit trail of the workflow execution, and is used by clients as a server side verification protocol.

¹¹It was adopted by W3C in 2002

8.4.2 Common Schema Elements

An advantage of the XML Schema Language is its modular design, which allows an element specification to be reused in other element specifications (in the same or in another schema).

The container schema includes two type¹² definitions, that are used in almost all element definitions in the schema:

- `roleType`
- `metadataType`

The `roleType` is used to create elements that represent an identity, and the `metaDataType` is used to create a descriptor (an element describing another element).

8.4.2.1 `roleType`

The primary function of the `roleType` is to wrap an X.509 certificate or a DN that represents an identity. The X.509 is wrapped in a `<X509Certificate>` element, that is type defined to be base64 text encoded binary data. The `roleType` also contains a `<person>` element containing name and e-mail address of the identity.

Specifying just a DN is adequate identification, but assumes that the recipient has another way of obtaining the corresponding certificate.

8.4.2.2 `metaDataType`

Meta data enables automated content processing, and this type is used to create elements that describe other elements or part of a tree. The container schema allows an element of `metaDataType` to appear as descendant of any *element* specification in the XML Schema¹³.

The `metaDataType` consists of three parts

- A free text element
- An enumerated type, that can be used as an application domain specific controlled vocabulary
- A hook to the Dublin Core namespace

The following shows the definition of the `metaDataType` using a graphical notation¹⁴:

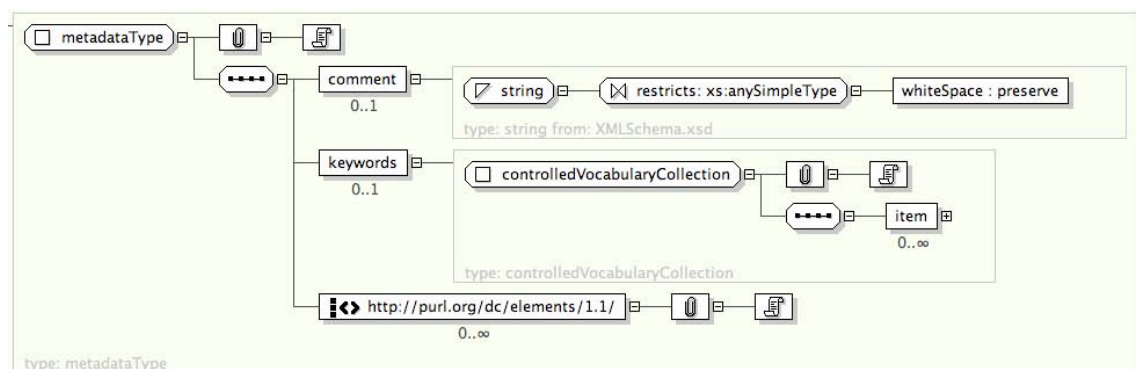


Figure 8.4.1 Definition of `metaDataType`

Each part represent a different level of structure, and all parts would normally not be required for all occurrences of an element of this type (`metaDataType`). A “free text” element to allow

¹²In an XML Schema, an element may be specified directly, or it may be *instantiated* from an existing type specification

¹³That is, all elements that are directly specified also includes a meta data element with a multiplicity of (0 .. 1)

¹⁴All graphical schema elements have been created using `xs3p` from <http://titanium.dstc.edu.au/xml/xs3p/>. The full schema documentation generated based on the documentation in the schema itself, is included on the CD-ROM

user provided feedback, a controlled vocabulary¹⁵ provides e.g. consistent categorization, and the Dublin Core allows automated processing.

The meta data type is mostly included as part of a generalized `commentType`, which is a union of the meta data type, and a `<commentFile>` element, which allows a file to be specified as a comment.

8.4.3 Server Endpoint (<info>)

The `info` element defines the server that should be used as part of the workflow exchange at a given point in time. The full specification is shown in the following figure:

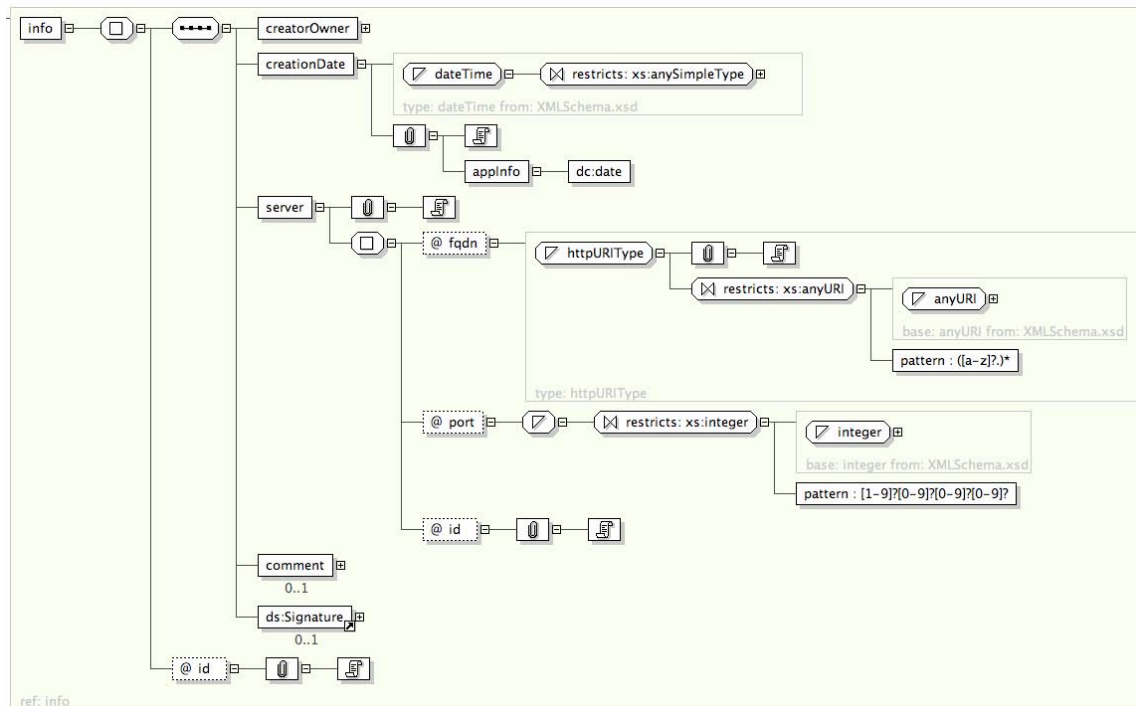


Figure 8.4.2 Definition of the `info` element

Note that the server is identified by its fully qualified domain name (FQDN), and despite being of `httpURIType`, it should not be prepended with `http://` (the value pattern restriction will also prevent this). Some URL data types will assume communication to be on port 80 (and to be HTTP), if the value is prepended with `http://`, but the schema should neither require HTTP¹⁶ nor that communication be on port 80.

The container is self-describing, and the server should not store any state, meaning that any implementation that is able to distribute the container to the next recipient, is able to process the container. Allowing a new server to be assigned uses this fact to provide the ability to use different servers at different points in the workflow.

The above concept could have been further extended to allow specification of multiple servers, as well as a prioritization, which would enable the client itself to handle fail over, if a server becomes unavailable. However, this feature has not been included in the schema.

To prevent a man-in-the-middle attack, the client must validate the signature on the `<info>` element, before this data can be used.

¹⁵The concept of a controlled vocabulary is implemented as an enumerated type and the schema must define the required words.

¹⁶Though HTTP will be chosen for communication as described in chapter 9

8.4.4 Workflow Specification (<workflow>)

The <workflow> element contains the actual workflow specification. A workflow comprises a number of activities that will be referred to as steps from this point forward. The schema workflow specification is based on a high level model that contains a number of <stepGroup> elements, with one or more step elements in each. The workflow specification creates the workflow graph by connecting a step with a <stepGroup> ¹⁷. The following figure shows this for a simple workflow comprising only sequence transition patterns.

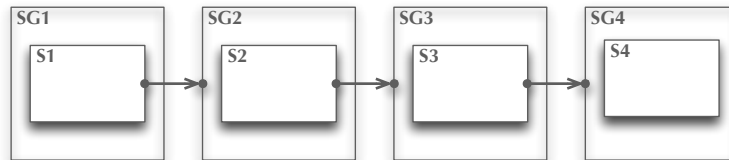


Figure 8.4.3 A high level model of a workflow

A <stepGroup> element can contain one or more step elements, and when execution reaches a <stepGroup> with $n > 1$ elements, all enabled activities will be executed, which enables the multi choice transition pattern. The synchronizing merge transition pattern is implemented by having the last step of each branch point to the same <stepGroup> . This design places the following requirements on the workflow specification tool:

- It must ensure that all (synchronously executed) branches contain the same number of steps (8.4-1)
- It must ensure that the final step of each branch point to the same <stepGroup> (8.4-2)

8.4.4.1 <stepGroup> Element Definition

A <stepGroup> specifies an organizational owner (orgRole), steps, outbound transition type (group type), and optionally the next and previous <stepGroup> . The definition is shown in the following figure:

¹⁷Not the other way around

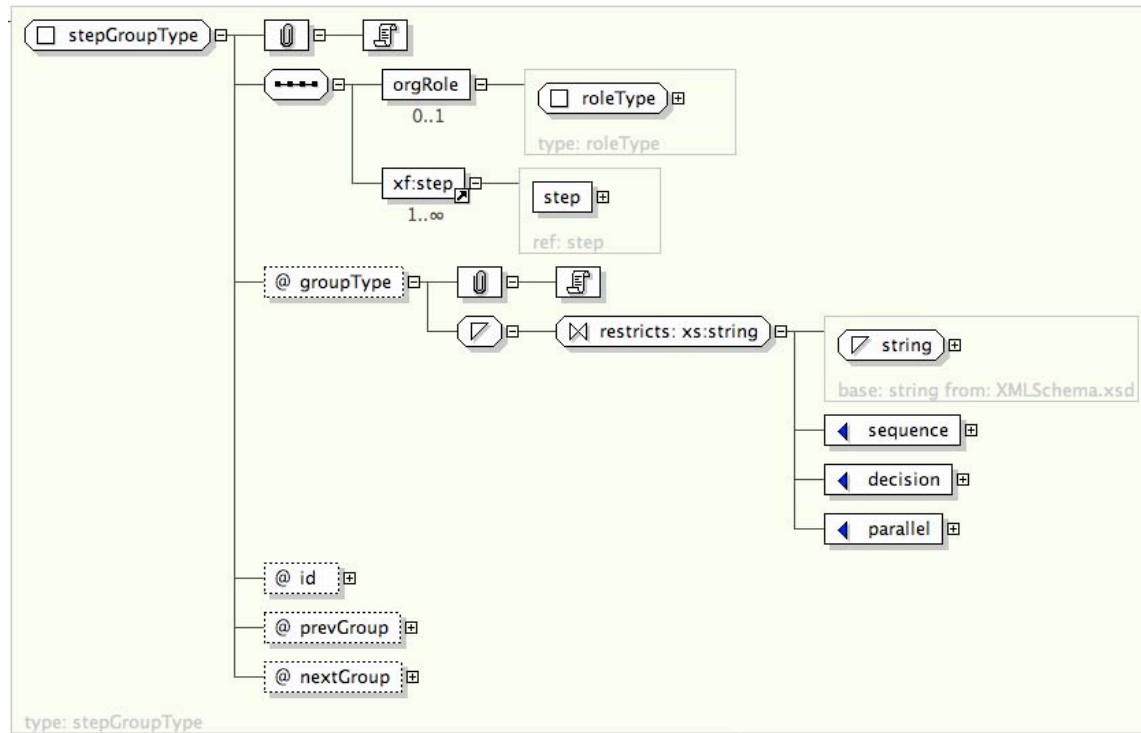


Figure 8.4.4 Definition of nextStepGroup Element

The `orgRole` is a `roleType`, representing an organizational owner of *all* contained steps, that should always have read access. The implementation can use this as a default *encrypt* to identity with `<stepGroup>` scope¹⁸.

If the next or the previous `<stepGroup>` can be determined before run-time they should be specified (using the attributes `prevGroup` and `nextGroup`) to make it easier to validate the workflow execution.

8.4.4.2 step Element Definition

A step must define the task (assignment and task), the assigned roles, and any other authorizations on this step. The schema assumes that only *one* assigned role will perform the step, and that only read authorizations are given. Finally a `step` must define the next `<stepGroup>`. The definition of the `step` element is shown in the following figure:

¹⁸Currently the schema doesn't include any alternative `<enc:EncryptedData>` elements for elements it should be possible to encrypt.

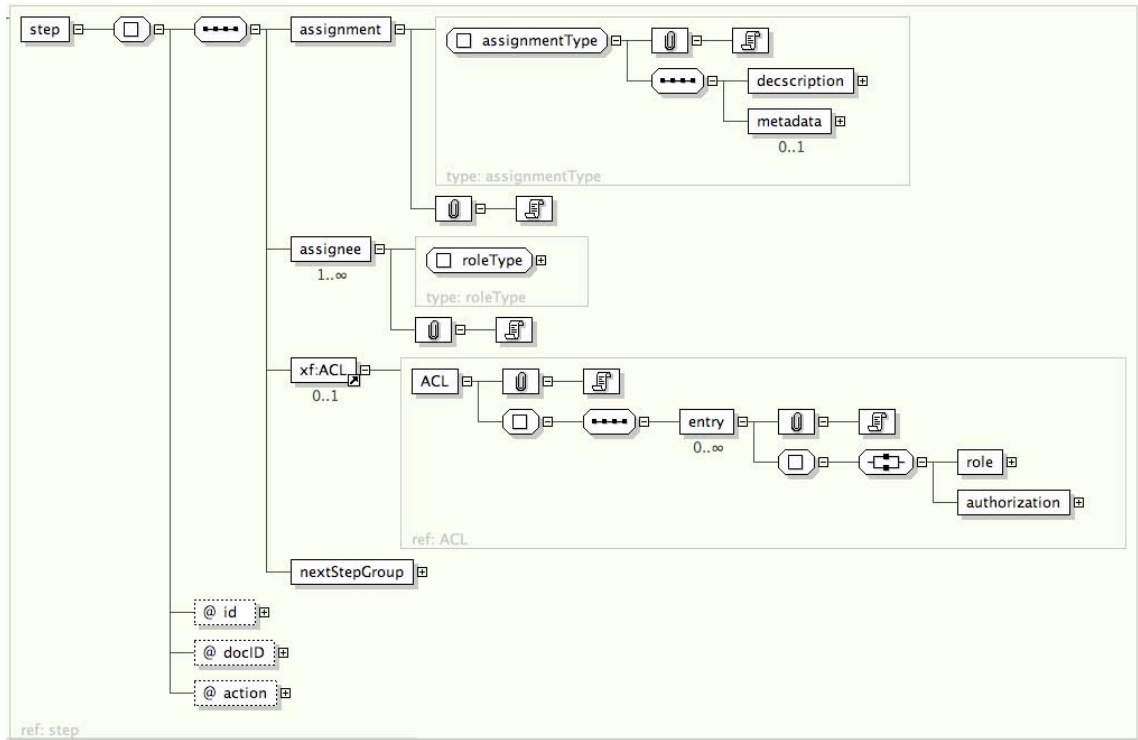


Figure 8.4.5 Definition of a `step` element

The `docID` attribute is described in 8.4.9.

8.4.4.3 Selecting the Next `<stepGroup>` Element

The `nextStepGroup` element that is a child of a the `step` element contains a specification of which `<stepGroup>` should be chosen next. The element implements two selectors for choosing a `<stepGroup>` `xml:id`. The following figure shows the element specification:

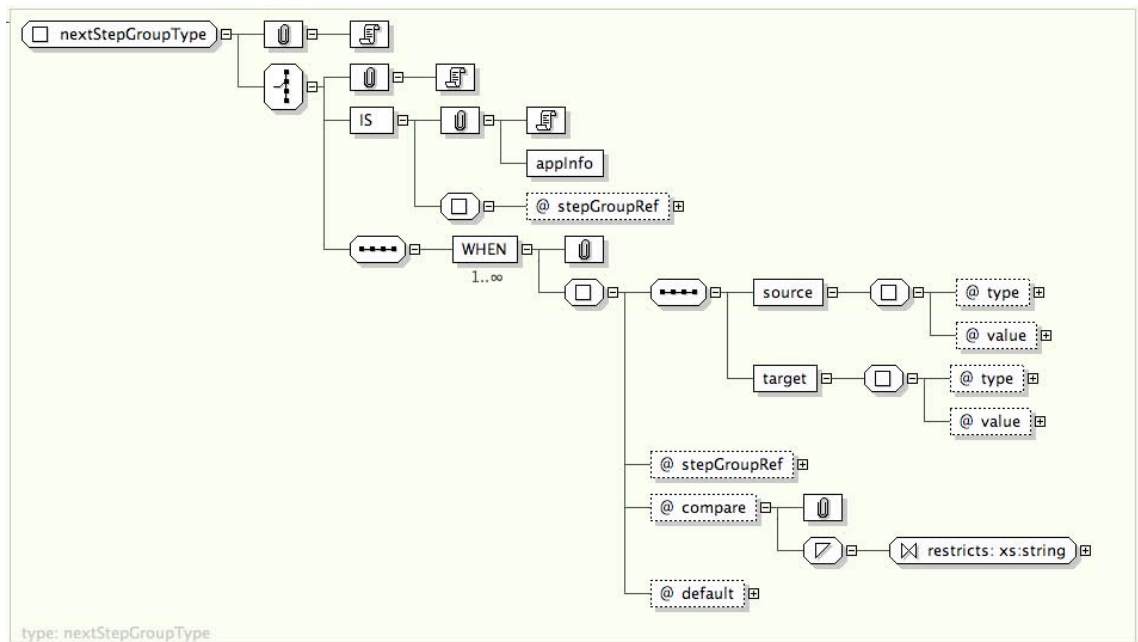


Figure 8.4.6 Definition of `nextStepGroup` Element

A `<stepGroup>` can be selected by direct reference which is done using an `IS` element, on which the `<stepGroup>` `xml:id` is specified as an attribute. This implements a sequence transition pattern.

The other option is to specify a number of tests, to be evaluated *with the container node as root element*. A test is specified as two XPath queries, and additionally the return value (string, int, and boolean) of each query can be specified, and how they should be compared (greater than, smaller than, equals). Return values and comparison is defined as an enumerated type. Finally it is possible to specify a default value, in case no test evaluates to true.

The schema design assumes that zero or one test will evaluate to true. If zero evaluate to true, the last default value seen should be used.

This mechanism for choosing a `<stepGroup>` implements an XOR-split, and an XOR-join is inherent in the specification.

The mechanism has two severe limitations.

- it lacks boolean operators to combine tests
- it requires detailed knowledge about the data to be tested, as well as its occurrence during run time

However, using XPath provides great expressive power, and is “free” because the XML parser or an external library provides this feature.

8.4.5 Document Versions (`<document>`)

Storing a document version requires storing the document, additional meta data, and the roles signature on the document. If a document was reviewed, the review categorization must be stored instead of the document. The following figure shows the definition of a `document` element:

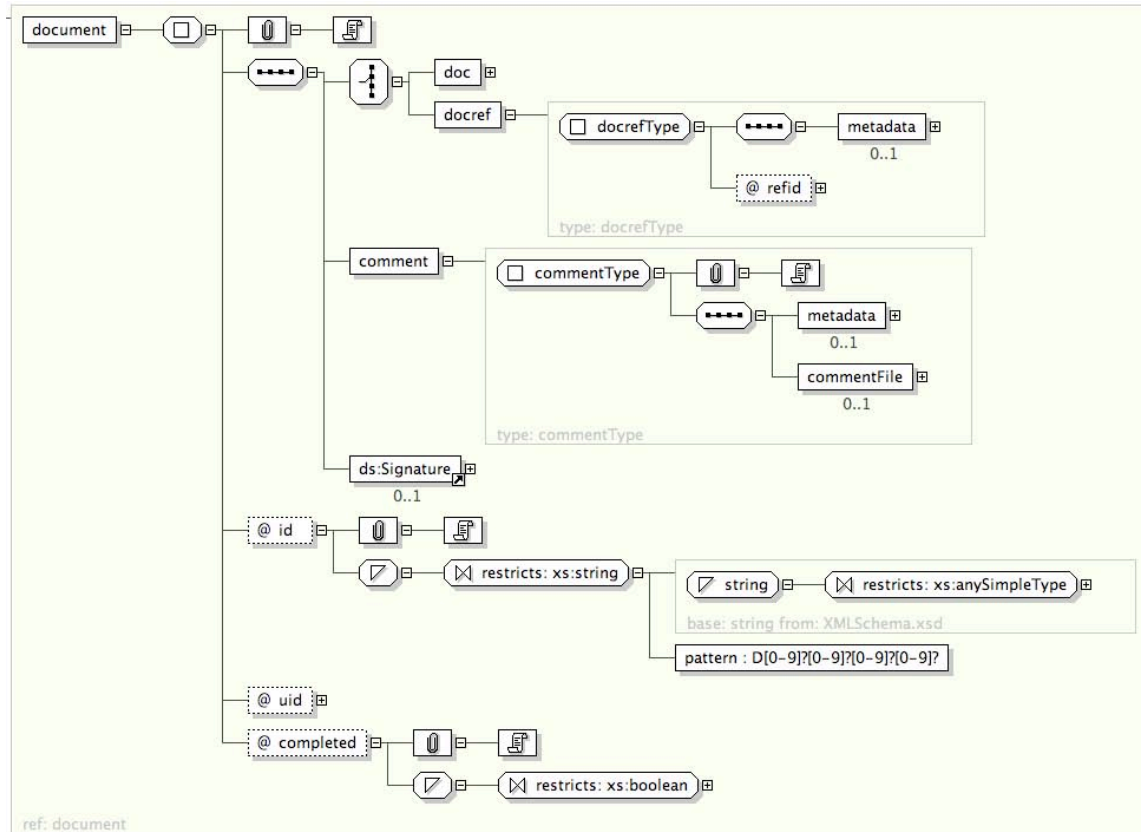


Figure 8.4.7 Definition of the info element

It allows either a document to be stored as base64 encoded data (the `doc` element), a review categorization which is stored as a `docRef` element that contains a reference (`xml:id`) to the reviewed document element, and the categorization is stored as a controlledVocabulary keyword in the `metadata` element.

The enveloped signature element could be made mandatory, but that would prevent the container from being distributed with a precreated structure of document elements.

8.4.6 Signature Protocol (<transactionlog>)

The `<transactionlog>` is a subtree to store `<receipt>` elements, which is shown below:

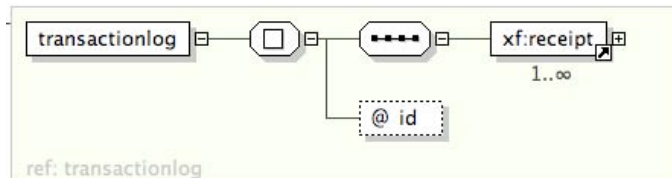


Figure 8.4.8 Definition of the transactionlog element

A `<receipt>` element is created by a server, when it has received a container from a role, and after it has successfully verified the container. The definition is shown below:

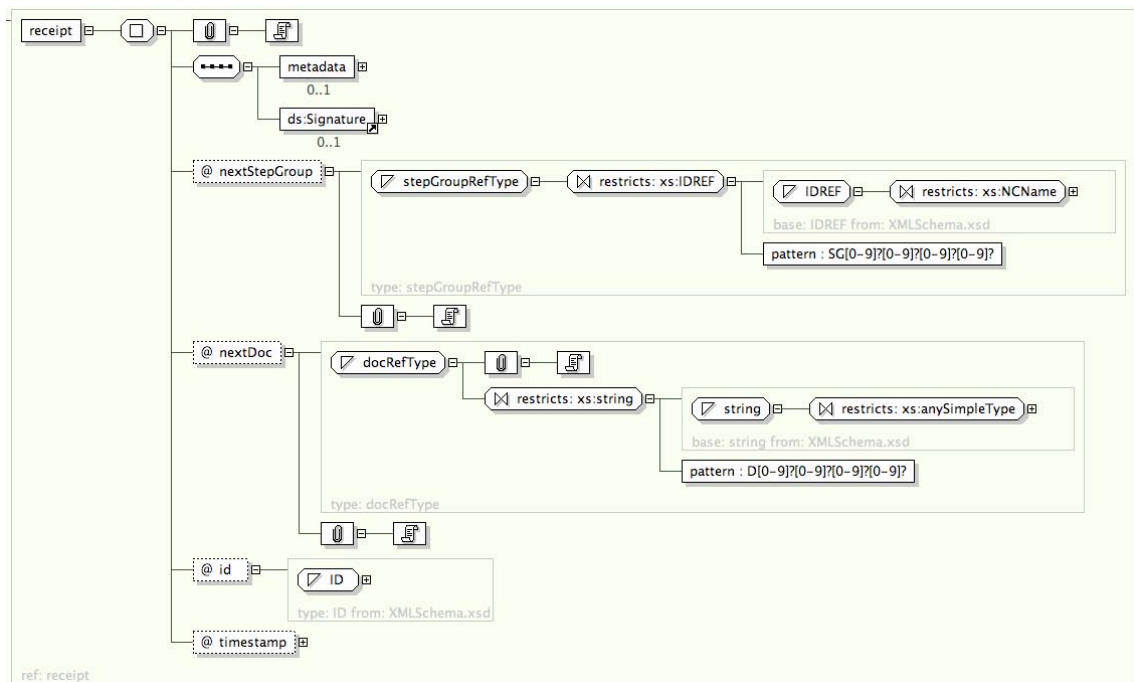


Figure 8.4.9 Definition of a receipt element

When the server validates a container, the `xml:id`'s of the next `<stepGroup>` and of the `<document>` become known to the server as part of the process. These values are included in the `<receipt>` to simplify the logic required by a client implementation.

A `<receipt>` element represents, that the container up until this point has been processed according to specification, and all signatures are valid (both parts must be verified by the server). Hence, when a client has validated the newest `<receipt>` element, the client knows that the current container is valid (provided it has been distributed to the client in a secure fashion).

The client can use this property of a `<receipt>` element as an incremental trust protocol, documenting that when the client has validated the current `<receipt>` the client has implicitly validated all previous versions. This means that to verify the client needs only verify the newest receipt element.

8.4.6.1 Element Signature Scope

Before any documents are added to a schema instance, an initial `<receipt>` element must be generated, and signed, which simplifies the verification algorithm because an initial state must not be handled.

The initial `<receipt>` element cannot be used to verify the same properties as subsequent `<receipt>` elements, because server verification includes verification of document signature, but no document versions have been added at this point. Thus, the signature on the initial `<receipt>` element must include a reference to the `<receipt>` element and the `<workflow>` element. It should *not* include a reference to the `<info>` element as this may change during the workflow execution, which would invalidate any referencing `<ds:Signature>` elements.

All subsequent `<receipt>` elements should include a reference to the newest document, in addition to the `<workflow>` and `<receipt>` element references.

8.4.6.2 Verification Protocol

The `<receipt>` elements in the `<transactionlog>` can be used to perform the following verifications on a container:

- Has the workflow specification been changed
- Has the current document been changed
- Has the workflow been executed correctly

By verifying the newest `<receipt>` element these properties are implicitly verified for all previous steps in the workflow, because a `<receipt>` element is dependent on the previous `<receipt>` element.

8.4.7 Controlling Attribute and Element Values

The XML Schema Language provides a number of ways for controlling the values that are assigned to an attribute or an element, including:

- Limited type system
- Regular expression
- Limited XPath queries
- Enumerated list

The container schema uses all controls except XPath queries to control element and attribute values. An `xs:restriction` based on an XPath query cannot look outside the scope of the subtree bounded by its (nearest) ancestor, and for XPath queries to be effective they must be able to “look” between the `<workflow>` and `<documents>` elements.

Because element `xml:id` values are used extensively, most `xml:id` attribute definitions are bounded by an `xs:restriction` that limits the allowed namespace to make it predictable, and to prevent namespace clashes.

8.4.8 Schema Namespaces

The schema is designed using the W3C namespace specification. All elements defined in the schema are named in the namespace `http://strandbygaard.net/xml/container`. The schema also incorporates elements from the following namespaces, and these must be declared in the root element of a schema instance.

- `xmlns:dc="http://purl.org/dc/elements/1.1/"`
- `xmlns:ds="http://www.w3.org/2000/09/xmldsig#"`
- `xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"`

To prevent namespace clashes, all elements must be prefixed using prefixes assigned to the relevant namespace, and all elements defined in the *X-flow* namespace will be prefixed with `xf`. In the following, if no prefix is specified, it will be assumed that the element belongs to the *X-flow* namespace unless otherwise specified.

8.4.9 Sealing the Container Structure

A `step` element must contain an attribute with the same properties as an `xml:idref` (but without the existence requirement on the referred `xml:id` value), to properly seal and enforce the container structure.

The subtrees `<info/>`, `<workflow/>`, `<document/>`, and `<receipt/>` are sealed by direct signature reference, but the link between the workflow specification and the added document

elements does not mean that any number of `document` elements can be added, and that an XML parser will still accept the container. This is undesirable, as it makes it possible to inject large quantities of data into the server with adverse impact on system resources.

The solution to this is to specify a `docID` attribute on a step that specifies which `xml:id` should be assigned to the corresponding `document` element. That makes it possible to perform a validation using a SAX based parser, that does not parse an XML instance into memory.

8.5 Creating a Container Instance

Creating an instance from the container schema involves the following steps:

- Setting the content of the `info` element
- Specifying the workflow as a production of `<stepGroup>` elements
- Producing an enveloped signature on the `info` and `workflow` elements respectively
- Creating the first `<receipt>` element
- Producing an enveloped signature on the `<receipt>` element, that also includes a reference to the `workflow` element

This section describes how a workflow specification is created, and how `<receipt>` elements are added. The examples are based on a workflow with two steps and a sequence transition in between. This section uses a number of code listings to demonstrate e.g. scope and signature references, and because XML tends to become rather verbose, the listings have been reduced to just the relevant elements and attributes, hence none of the code listings are schema valid.

The schema itself contains documentation of each element or attribute and its usage, but the schema specification also requires that a number of elements reference each other correctly. These references cannot be checked by the XML parser because they are either not known at a given point in time, or because no query can reach between a reference and the referenced `xml:id`. The following sections shows the basis for validating these dependencies.

8.5.1 Creating a Workflow Specification

The following listing shows how a workflow specification is constructed from `<stepGroup>` elements using `xml:id` and `xml:id` types.

The specification snippet contains two activities (steps) with a sequence transition connecting them. Mapping this to the high level model shown in figure 8.4.3, this results in a container with two `<stepGroup>` elements (SG1, SG2) each containing one `<step>` element (S1, S2).

A step must reference the next `<stepGroup>` element in the `<nextStepGroup>` element, and if a step is the final step (as shown in the listing) of a workflow, that step must reference its parent `<stepGroup>` element.

```

<xf:container>
  <xf:info id="info"/>
  <xf:workflow id="workflow">
    <xf:stepGroup id="SG1">
      <xf:step id="S1" docID="D1" action="modify">
        <xf:assignment/>
        <xf:assignee>
          <xf:DN/>
        </xf:assignee>
        <xf:nextStepGroup>
          <xf:IS stepGroupRef="SG2"/>
        </xf:nextStepGroup>
      </xf:step>
    </xf:stepGroup>
    <xf:stepGroup id="SG2">
      <xf:step id="S2" docID="D1" action="finalize">
        <xf:assignment/>
        <xf:assignee>
          <xf:DN/>
        </xf:assignee>
        <xf:nextStepGroup>
          <xf:IS stepGroupRef="SG2"/>
        </xf:nextStepGroup>
      </xf:step>
    </xf:stepGroup>
    .....
  </xf:workflow> ..... Workflow Specification
  <xf:documents/>
  <xf:transactionlog/>
</xf:container>
  
```

8.5.2 Adding the Initial <receipt> Element

Before the initial `<receipt>` element can be added, the `<info>` and `<workflow>` elements must be signed, by adding an enveloped signature to each element.

The `<receipt>` element must include a reference to the next `<stepGroup>` (in the listing that is SG1), and the enveloped signature must contain references to the `<info>` and `<workflow>` elements.

```

<xf:container>
  <xf:info id="info">
    .....
    <ds:signature>
      <Reference URI="#info"/>
    </ds:signature>
  </xf:info>
  <xf:workflow id="workflow">
    ..id="SG1"
    <ds:signature>
      <Reference URI="#workflow"/>
    </ds:signature>
  </xf:workflow>
  <xf:documents/>
  <xf:transactionlog>
    <xf:receipt nextStepGroup="SG1" id="R1">
      <ds:signature>
        <Reference URI="#R1"/>
        <Reference URI="#workflow"/>
      </ds:signature>
    </xf:receipt> ..... Initializing receipt
  </xf:transactionlog>
</xf:container>
  
```

8.5.3 Adding a Subsequent <receipt> Element

The first <receipt> element that is not the initial, is added the first time a role submits a container, and to separate the individual actions, the listing differentiates using three colors, representing three different points in time. Blue represents the initial <receipt> element, green represents the additions made by a role, and red represents a subsequent <receipt> element.

The initial <receipt> element points to the first <stepGroup> element (SG1), and the user adds a document with `xml:id` (D1).

The signature in the subsequent <receipt> includes a reference to the <workflow> element (`#workflow`), the previous <receipt> element (`#R1`), and the new <document> element (`#D1`).

```

<xf:container>
  <xf:info id="info"/>
  <xf:workflow id="#workflow">
    <xf:stepGroup id="SG1">
      <xf:step id="S1" docID="D1">
        .....
      </xf:step>
    </xf:stepGroup>
    <xf:stepGroup id="SG2">
      <xf:step id="S2" docID="D1">
        .....
      </xf:step>
    </xf:stepGroup>
  </xf:workflow>
  <xf:documents>
    <xf:document id="D1">
      .....
    </xf:document>
  </xf:documents>
  <xf:transactionlog id="transactionlog">
    <xf:receipt nextStepGroup="SG1" id="R1">
      .....
    </xf:receipt>
    <xf:receipt nextStepGroup="SG2" id="R2">
      <ds:Signature>
        <ds:Reference URI="#R1"/>
        <ds:Reference URI="#workflow"/>
        <ds:Reference URI="#D1"/>
      </ds:Signature>
      ..... Subsequent Signature
    </xf:receipt>
  </xf:transactionlog>
</xf:container>

```

9

Secure Workflow Model

This chapter describes the design of a prototype client-server application, that satisfies the requirements stated in chapter 7 and is based on the data model described in chapter 8.

9.1 System Architecture

The system is designed as a 4-tier application with the following tiers:

Client Application Is used by a user to work with a container. This application must be executed on the user's computer

Web Interface Is used by users to retrieve waiting containers, and by a system administrator to control the system

Application Server Is the server endpoint specified in a container

Database Server Is back end data storage for the application- and web-server.

From a user's point of view the client application and web interface is simply the *client interface*.

9.1.1 Client Interface

The entire *client interface* is designed to be fully distributable through most standards compliant browsers¹ that support XHTML and CSS.

A user is required to sign a document which requires access to the user's private key, and at least using OCES certificates, that *requires* that the private (signing) key is only available to the user², meaning that the signing operation must be performed by an application that is run on the users computer.

The client application that is run on the users computer is implemented in Java, and can be executed either as an applet or as a standalone program. This application is also used to submit a container to the server, once it has been signed.

9.1.1.1 Certificate Access

The scope of this thesis is not to provide a method for achieving secure access to a private key from a user account on an operating system. The problem with *secure* access to a private key,

¹Most browsers support *their own* plugin-in model for extending the browsers functionality (Mozilla supports XUL extensions, and Internet Explorer supports ActiveX). In the following, when referring to browser functionality it will be assumed that this functionality is available in *at least* Firefox, Internet Explorer, and Konqueror.

²This is a requirement both of the OCES certificate policy, and TDCs terms of use

e.g. in the form of a ASN.1 structure (a PKCS12 file), is that the existing OS security controls are deemed insufficient to protect the private key, as it requires *extra* security measures.

As a security failure concerning a private key may potentially have extensive effects, it can be argued that the key requires a higher level of security, than does many other parts of the system.

Most solutions tend to provide more of a protective measure against the user, than against real external threats, or at most another layer of complexity separating an attacker. A security component may enforce a password protection of a private key, but an attacker can just as easily modify the actual security component, thus completely circumventing the protection of the certificate; this attack would require circumventing OS access control, which is no different than reading a certificate protected by OS access control mechanism.

Hence the actual problem of secure access to a certificate arises because most operating systems, or indeed most hardware platforms do not offer a *managed security architecture*.

The *X-flow* system is designed to allow access to standardized certificate devices, which includes PKCS12 files, and a simple extension mechanism for adding support for other devices (e.g. PKCS11 interfaces).

Other solutions such as directly³ supporting TDCs cryptographic component, or the OCES CD-Card, are not supported.

9.1.2 Client-Server Communication

The client application communicates with the server using the HTTP or HTTPS protocols (depends on webserver configuration).

Choosing a client-server protocol that either communicates on an “uncommon” port, or that uses a port typically assigned to another protocol, will prevent the client-server protocol from traversing firewalls that use a default deny policy, or firewalls that incorporate application protocol level proxies.

Even though other protocols are more efficient than HTTP, it is the “ubiquitous Internet protocol”, and will typically always be open.

9.1.3 Application Server

The application is designed as a standalone Java program. Several frameworks for creating Java server applications exist (e.g. servlet, EJB, or JSP), but the system does not require most of the (mostly web application related) functionality offered by these frameworks, so there is no point in using them.

This means that the application server uses a fairly light webserver that is not intended for large numbers of transactions, but given that the application must also parse XML files that are potentially > 1Mb, the webserver will not be the component that cannot scale. Should the internal webserver become a problem, the design is easy to implement as a servlet.

Any number of programming languages can be used to implement the application server, but Java has the advantage that it combines a number of properties, that are hard to find all in another programming language. These properties include:

- Freely available library supporting the XML Signature standard (Apache XML Security)
- Robust XML parser with good support for XML Schema (Apache Xerces)
- Executed in a virtual machine (JVM)

³The cryptographic component distributed by TDC is both accessible through Microsoft CAPI, and through its own API.

Both Apache projects are also available for C, but C is not executed in a VM and is inherently prone to e.g. buffer overflows which cannot occur in a Java application.

9.1.4 Database

As database back end for the application server, the system uses the MySQL database. The system is designed to use version 4, but version 5 can be used without any modifications.

The database schema comprises 5 tables:

Containers Stores all containers known to the system. When a user is presented with a list of waiting containers, that is simply a filtered listing of this table.

Users All users known to the system.

Roles All roles created in the system

Permissions All permissions assigned to roles

RoleAssignment Contains an entry for each user that is assigned one or more roles. Currently, a user can only be assigned 5 roles, but this can easily be extended by extending the database schema

In this case the database is simply a location to put containers while waiting for user interaction, and two other alternatives could have been either storing it as a file on the OS file system, or using an embedded engine such as Berkeley DB.

Storing the containers on the OS file system is not an option because that only allows searching file names, and searching by role is required to retrieve a container assigned to a role ⁴.

Using the Berkeley DB API generally requires less administration, but prototyping using a relational SQL database is faster, so MySQL was chosen.

9.2 XML Programming Model

The *X-flow* system uses the Document Object Model (DOM) API for working with XML files (including containers), and this API is used both by the client application, and the application server.

This choice is largely made because, that it makes working with XML signatures much easier, as the chosen library also uses DOM. This means that DOM objects can be passed between XML Signature implementation, and the client application or application server.

The main disadvantage of using DOM is that it loads the *entire* XML file into an object in memory, which means that working with large (> 2Mb) requires a lot of memory. It also means, that this design will not work well on most mobile devices, as they don't have the necessary memory.

Other (user interactive) applications working with large XML documents (e.g. OpenOffice or Microsoft Office) only parse fragments of the DOM and use a combination of DOM and SAX to achieve the best properties of the two, however *X-flow* won't integrate these optimizations.

9.3 User Interface

The client interface (client application and web interface) together implement all identified use cases. Screen shots of all screen are included in appendix D (client application) and E (web interface).

⁴The file name could be used to store e.g. the role's name and other data that would then be searchable, but this kind of (mis)use of namespaces should be avoided.

10

Model Implementation

This chapter describes how major subsystems have been implemented, and it is organized to match the steps a container passes as part of processing.

This chapter does not include a description of each class in the implementation, rather the source file of each class provides a description (in Javadoc) of the functionality of that class. This documentation is included in `$CDROM/javadoc`.

10.1 Accessing the Container

Accessing the container is done differently in the client and in the server.

A user only performs a limited number of interactions with the client application as part of submitting a container to the server, hence the client application uses a file oriented approach in which the container is kept in memory a little as possible. The client application will flush the DOM back to disk as soon as possible when a change has occurred.

The server on the other hand *only* works with the container as a DOM in memory, until the DOM has been serialized and stored as a BLOB in the database.

10.1.1 Parser Configuration

Both the client and server application use the Apache Xerces XML parser. The client wraps the parser in the class `net.strandbygaard.xflow.container.XmlParserFactory`, that implements the `XmlParser` interface in the same package.

This factory class ensures that a parser object is correctly configured, which includes:

- Namespace support
- URIs of all namespaces that will be used
- Schema validation

The schema is part of the model specification, but it is not always necessary to load a container instance using a validating parser. When a client receives a container the `<receipt>` has to be verified to ensure that the container is valid, hence no (parser) validation is required.

The client *should* validate the container before signing a document to ensure that the user has not added any invalid data, but since a user can circumvent this check, it is not strictly required.

The server *must* validate the container before further processing, but once a container has been validated it need not be validated again unless it has been outside the control of the server.

The server configures instances of `XmlParserFactory` to be validating and contains another implementation (`XmlQuickParser`) of the `XmlParser` interface, that creates a non-validating parser.

Using a validating parser to verify the XML container is a an easy approach to validating an XML file, because it is implemented by the parser, but parsing an XML file with validation is generally *much* slower than just ensuring that the file is well formed.

10.1.2 Loading A Container (Client)

The server works with the DOM as a primary object, but in the client it is necessary to maintain a session state in order to quickly present the user with information from the container (instead of having to retrieve much of the same information several times).

To do this the client is implemented as a two level object hierarchy, that must be populated (`container` object must know about the container, `workflow` object about the workflow and so forth) when a container is loaded.

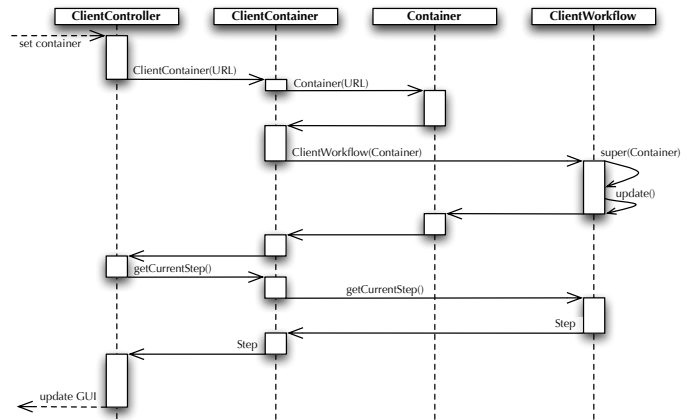


Figure 10.1.1 Simplified process of loading a container in the client

Figure 10.1.1 shows the lower part of this hierarchy. This part can be treated as the call path or routing layer, which the invocations also suggest. The upper hierarchy (not shown) comprises a super class to each of the classes, and in this layer most of the actual processing (e.g. extracting data from the DOM or creating helper objects) is performed.

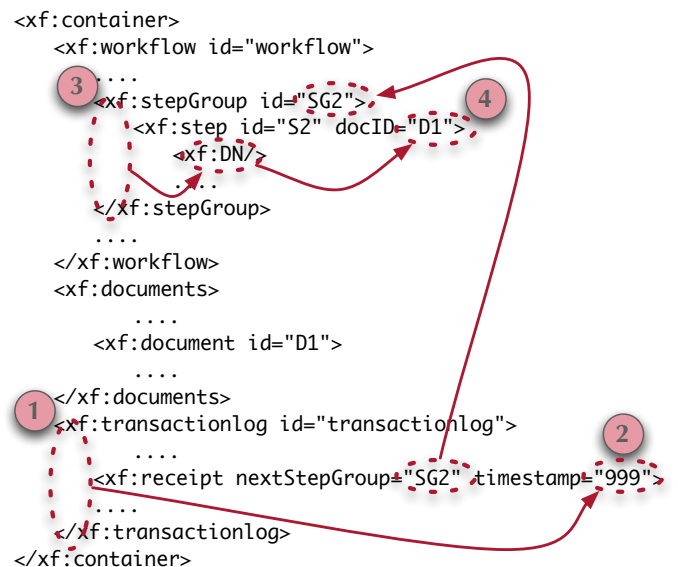
10.2 Workflow Engine

10.2.1 Selecting the Current Step (Client)

In order to select the current step (to find the `xml:id` of the new `<document>` to be added) the client must find the current `<stepGroup>`, and the `xml:id` of this is included in the newest receipt, which is found by iterating over the `<transactionlog>`, and selecting the `<receipt>` with the newest time stamp.

When the current `<stepGroup>` is found, it is possible to iterate over the DN of the assignee in each step in the `<stepGroup>` and select the one, that matches the current user. This implies, that two steps in a `<stepGroup>` cannot be assigned to the same user.

The code listing shows how this algorithm works on a container instance. All direct



descendants of the `<container>` always¹ have an `xml:id` and this can be used to quickly select each of the elements in the DOM using either an XPath query (such as `//*[id='workflow']`) or the `getElementById()` from the `org.w3c.dom` API.

10.2.2 Verifying the Current Step (Server)

The server must perform the same process as the client in order to verify a container that has been submitted, however the server must perform 2 additional steps:

- Verify the signature on the added `<document>`
- Verify that the assignee specified in the current step is the signer

10.2.3 Multi choice and -merge Container (Server)

The multi choice pattern has not been fully implemented due to lack of time. The implementation does include a full framework to support multi choice pattern, and only one class (`net.strandbygaard.xflow.server.Selector`) needs to be implemented.

The multi choice transition is implemented using the concept of selectors described in section 8.4.4.3, and the class that needs to be implemented, is a class that takes an expression as argument and evaluates this on the container DOM.

The multi merge has not been implemented due to lack of time nor does the code base include a skeleton framework, the following, however, describes the implementation principle.

To merge the branches of execution, the added `<document>` and `<receipt>` elements of each branch must be included in the last submitted container, which is done by parsing each container and copying the `org.w3c.dom.Node` objects into the newest container. Doing this will not break the signatures because the entire subtrees referenced by `<document>` and `<receipt>` element signatures are copied.

Finally, a new `<receipt>` is generated in the container to which all the other elements were copied. The signature on this receipt must include all added elements in its scope.

10.3 Certificate Access

Both client- and server applications and support tools only support X.509 certificates, that are stored in a PKCS12 file or which can be loaded from the container document.

While an X.509 is a platform independent standard, many certificate storage devices (Aladdin eToken, Java Keystore, Microsoft CAPI) are typically dependent on both platform and implementation. A PKCS12 file is a platform independent storage container for X.509 certificates representing the *broadest common denominator*, and as the time available for this project didn't allow implementing support for several certificate devices, this device was chosen.

This device is the standard way of supporting certificates in a server application (e.g. BEA WebLogic, or IBM WebSphere), where either PKCS12 files are used or DER/PEM encoded key and certificate files².

However, in a client application, this will frequently be inadequate. Generally, a client application will be required to support PKCS12 files (if only to facilitate testing), as well as a more use case specific method (e.g. Microsoft CAPI), and section 10.3.1 describes the client application's support for OCES certificates and certificate devices in general.

¹The value is declared final in the container schema

²That may easily be converted to a PKCS12 file using e.g. OpenSSL

10.3.1 Certificate Device Plugin Architecture

Certificate support has been implemented using a plugin architecture that allows extending device support to new devices. The primary intention of this architecture is to make it easy to add a class implementation, that uses the Java Native Interface (JNI) to access a platform dependent PKCS11 compliant drive, e.g. CAPI or the driver for a cryptographic token.

As such the device plugin architecture isn't implemented with support for loading and unloading device implementations at run-time or choosing a device implementation at run-time. This means that a client built from the current code base only can be build with support for one type of certificate device.

The OpenSign applet from the OpenOCES project [58] is an application comparable to the *X-flow* client, given that it is implemented in Java and uses X.509 certificates to create XML Signature. This applet includes a plugin architecture for dynamically loading certificate device implementations.

10.3.1.1 System Implementation

In the client, the certificate device plugin architecture is implemented using a simple *Factory* pattern³, to separate client code implementation from the device implementations. This relationship is shown in figure 10.3.1.

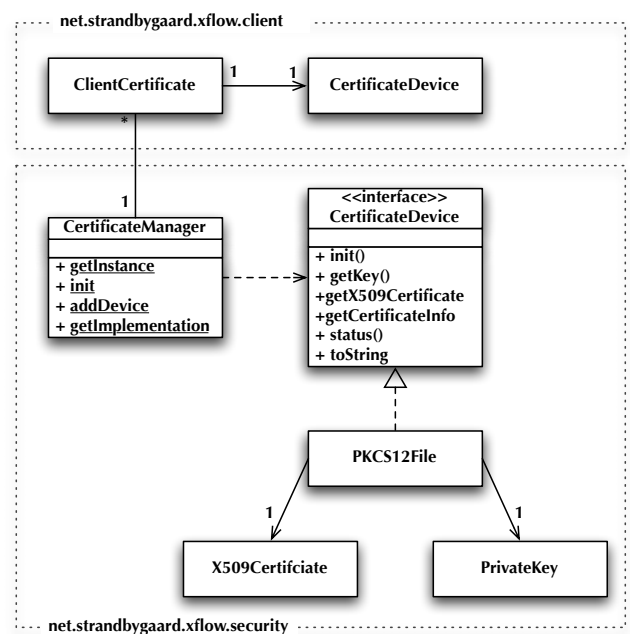


Figure 10.3.1 UML class model of client certificate support

³The implementation is combination of the *Factory*- and *Abstract Factory* patterns defined in [59].

10.3.1.2 Extension Points

The client code base is provided with a limited number of extension points, that can be used to add support for new certificate devices. Currently the code base contains two levels of hooks:

1. Support for *statically* binding to a class that implements the interface `CertificateDevice`
2. GUI support (including state) for choosing a certificate device at run-time (only applies to the client)

Obviously, (1) should be re-implemented using a class loader to load an implementation at run-time.

The client is built on the Model View Controller (MVC) pattern, and dynamic certificate selection has been fully implemented in the View part and partially in the controller. The menu item `Settings -> Certificate` is used to choose the certificate device, and has been populated with three example choices (see figure 10.3.2).

The actual choices should be added, as handlers for the certificate devices are registered, but this has not been implemented yet. Instead, the example certificate devices have been added.

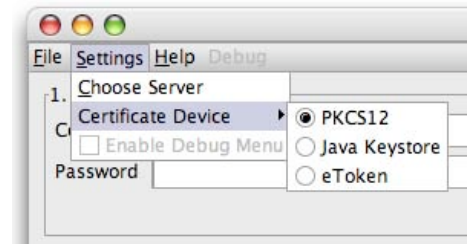


Figure 10.3.2 Selecting a certificate device

10.4 Signing and Verification

Signing or verifying an XML Signature requires that the XML parser is able to resolve all required URIs, including:

- Element references
- Signature properties (e.g signature- and canonicalization algorithms)

Only document local element references are used (of the form `#<reference>`), and they can always be resolved. The implementation assumes that either (1) HTTP is available to resolve other URIs or (2) a catalog for resolving references has been configured.

10.4.1 Signing an Element

A signature is generated as an enveloped XML Signature on an element URI, and the class `net.strandbygaard.xflow.security.XmlSigUtils` implements an API, that is used by the client application to this signature.

To make the client application more robust to errors (e.g. the user unexpectedly shuts down the computer), the implementation serializes the container DOM to disk before and after the signing operation using a safe overwrite method (`XmlUtils.saveDomAsFile()`) ensuring that the original container file is not overwritten before the new one has been successfully written to a temp file (and the temp file is not removed before the old file has successfully been overwritten by the new).

The implementation does not check the basic usage specified in the supplied certificate, so any certificate can be used. The client *should* check basic usage and adhere to criticality, but a recipient *must* do so, hence this has not been implemented.

10.4.2 Verifying a Signature

The class `net.strandbygaard.xflow.security.XflowTrustAPI` implements an API that is used by the client application to verify an XML Signature. This approach removes dependencies in the rest of the code to the Apache XML Security library, and it provides a unified API for

resolving certificate trust (the method `XflowTrustAPI.verifyCertificate()`). The server implementation has an analog implementation⁴ in `net.strandbygaard.xflow.server.ServerSigUtils`.

In the implementation, a user certificate is trusted if the issuing CA is trusted and no CRL check is performed on a user certificate.

10.5 Client-Server Communication

The client application communicates with the server using XML-RPC over HTTP⁵. This protocol has two benefits:

- Compared to other message protocols (e.g. SOAP) it is very simple
- It provides a programming paradigm that is very similar to that of Java
- It allows late binding of the session

An XML-RPC endpoint is invoked with a single method, and it provides support for most standard data types (including `String`, `int`, and `byte[]`). As the container file must be parsed by the server, sending the container to the server as `byte[]` is the best way to transfer the file.

Late session binding is important to prevent denial of service attacks based on resource starvation. If the server allocates many system resources by creating session objects etc. before authenticating a user, this opens the server to DoS attacks.

Using XML-RPC over HTTPS it is possible to authenticate a user through the SSL connection by requiring a client side certificate when performing the SSL handshake⁶, which also makes authentication “transparent” to the user.

However, using XML-RPC and SSL does limit how the systems can be deployed, because the protocol messages cannot be routed and a direct (point-to-point) connection to the *X-flow* application server is required (e.g. proxying the SSL will *not* work). Lack of message routing is currently less of a problem, but with the advent of the much touted *services oriented architecture* (SOA), this will become a hindrance.

Lack of SSL proxying support will not affect client deployment, as few upstream connections employ such proxies⁷, but it affects how the server is deployed because it *is* common to proxy incoming SSL connections early on, so the content is exposed to existing intrusion detection systems (IDS) and similar systems.

Using SOAP and the Web Services security standard (WS-security) would overcome both limitations because it wraps the security in the SOAP message, and message routing is addressed with WS-routing [60].

10.6 Audit Trail

The code base `net.strandbygaard.xflow.*` is instrumented with logging using the Apache Commons Logging project (`org.apache.commons.logging`), and the system has been tested with the log facility, `log4j`, though any facility that implements the interface should work.

⁴The client and server methods must behave slightly different, and will have different parameter requirements, but it was speedier to just duplicate the 30 or so line of code.

⁵XML-RPC can be used over just about any communications protocol that allows sending and receiving plain text messages (e.g. FTP and SMTP)

⁶Websites can typically not require a client side certificate in the handshake, because the user cannot be expected to have a certificate, or because the website doesn't have a registration between the user and the certificate. However, in the *X-flow* system a user will always have a certificate, and the server will always have a registration matching the user and the DN in the certificate

⁷They obviously “break” the security of the SSL connection from the clients point of view.

The audit trail (log statements) is generated in 3 levels (debug, info, and error) and each level can be selectively enabled or disabled on a class-by-class basis⁸.

Only the info and error levels contain information relevant to the audit trail. The info level is used to log check points in transaction processing, as well as the result of relevant operations. The error level is used to log all application states, that results in the current execution being halted (e.g. that a signature is invalid).

Log4J allows different methods for capturing log lines. The current project logs all messages generated by the client and server to the console. This is practical during development, but in a production setup logging should at least be done to a file, and preferably to a different computer (e.g. a central syslog facility). This is easily changed in the log4J configuration file.

10.7 Web Interface

System administration and download of containers is provided through a web interface that is written in PHP and runs on an Apache webserver.

The web interface provides a number of form based views that can be used by the administrator to add, update, or by a user to download waiting containers.

Before a user can access the web interface, the user must login. In a production environment, only login using a client certificate should be allowed, but in order to make testing easier, provision for a username/password based login mechanism has also been included⁹.

When a user logs in using a certificate (or username and password), the user is authenticated by looking up the presented DN in the `users` table in the database, which includes field storing the users DN.

The web interface is designed with two different user profiles: User and administrator, that determines if a user has access to the administrative part of the interface.

⁸The supplied log4J configuration file in `$CDROM/source-code/resources/log4j.conf` demonstrates this.

⁹Most browsers cache SSL session cookies until the program is closed, which means that to switch user on the site (during testing) it would be necessary to close and reopen the browser.

11

Model Analysis

This chapter provides an analysis of how well the system design and the implemented prototype satisfy the requirements stated in chapter 7.

11.1 Workflow Support

11.1.1 Supported Transitions

The following tables shows support each of the required transition patterns in both the design, and the implemented prototype.

Table 11.1.1: Feature Matrix			
Requirements Analysis		Feature Support	
Label	Property	Data Model	Implementation
7.2-7	Sequence		
7.2-8	AND-split	See (7.2-12)	
7.2-10	XOR-split		
7.2-9	AND-join	See (7.2-13)	
7.2-11	OR-join		
7.2-12	Multi Choice		
7.2-13	Multi Merge		

Table 11.1.1 The table lists support for transition patterns in the design and implementation.

The design is based on multi choice and multi merge instead of the simple transition pattern variants XOR-split and AND-join, and because support for these patterns are incomplete in the prototype the system cannot process workflow specifications that contain branches.

In the current design, as implementation of multi merge can possibly become quite resource intensive, as each the received container for each branch must be retrieve from the database and parsed (non-validating) in order to determine if all submissions, or an adequate subset, have been received. This design could be improved significantly if the application server maintained a separate state table for each ongoing branch for which it has received at least one submission.

Extracting the necessary information from the container while it has already been parsed, and aggregating this in a state table, would means that the server would not have to re-process all received containers for each submission in a branch.

11.1.2 Container Template Instance

The container schema specifies a specific DN for the assignee of each step, which means that a container *instance* is bound to the specified roles. Even using multiple assignees per step this approach is not adequately flexible if a single container *template instance* is to support a frequently occurring workflow process (e.g. one involving the customers of a company).

The web interface includes an authorization model that can be used to drive a *template instance* system in which the DN of an assignee is specified as a authorization role (the entries in the Roles table). When an actual workflow is to be started this template can be populated by selecting entries from the RoleAssignment table.

11.2 Model Checking using XML Schema Language

The *X-flow* system is designed to use a validating parser to verify the structure and contents the XML container, and this carries an overhead in terms of run time as validation is much slower.

The following table has been compiled with measurements from the client application (the client application also logs the parsing time).

Table 11.2.1: Parsing Time				
No. of Elements	Non-validating	Validating	File size	Java heap size
143 elements	32ms	3809ms	20kb	-
173 elements	32ms	3540ms	20kb	-
275 elements	32ms	3700ms	24kb	-
613 elements	32ms	3937ms	68kb	-
208 elements	834ms	4060ms	5.8Mb	min. 68Mb

Table 11.2.1 The table lists how parsing time and Java heap space is affected by number of elements and file size.

11.2.1 Parsing Time

As the table shows, neither element count nor file size has an impact no parsing time, with the only exception being the run time of the non-validating parser using a large file. For small file sizes the relative difference in run time is constant, and the gap decreases when parsing the large file.

Although there is a significant different in parsing time (non-validating parsing is approximately 100 times faster for small file sizes) it should be noted, that validating parsing is barely affected by file size and that the parsing time is not prohibitive, so as to be unusable.

The server currently spawns a separate thread for each container submissions, and having the thread block for 4 seconds waiting for the parser is not an optimal design. However, the server's XML-RPC request handler is changed to place all submitted containers on a "loading queue" from which the parser parses containers, and passes them to the current server threads.

Assuming that the usage pattern doesn't inhibit very large peaks, the additional processing would be acceptable¹, and due to the initial SSL handshake with mandatory client side certificate, it is not possible to use this application response (the long processing time) to perform a denial-of-service attack.

¹Many applications employ timers of a minute or more, waiting for subsystems to complete.

11.2.2 Memory Usage

The system is implemented using the DOM API and as this API loads an entire XML file into memory, it can use a lot of memory.

The small XML files could be loaded regardless of the amount of memory the JVM was started with. However, it was not possible to load the large XML file without allocating at least 68 Mb memory to the JVM. The test program used to perform the measurement performed not other operations.

This memory requirement will likely not pose a problem when using the client application on a modern computer (Java requires at least 64Mb to start, anyway), but it also compounds the fact that the server will not be likely to achieve multiple per second transaction numbers².

11.3 Security Analysis

The design and implementation presented in this report was intended to satisfy the security objectives stated in section 5.4, and the following table summarizes how well this was achieved.

Table 11.3.1: Security Level		
Label	Objective	Design/Implementation
5.4-2	A role <i>must</i> be authenticated to access the system	A role cannot access any part of the system without authenticating. To access the web interface or submit a container a user must present a certificate representing a user account.
5.4-3	A role <i>must</i> be authenticated to perform an activity	A container includes authentication data representing a user, in the form of a digital signature in the submitted container
5.4-3a	A role <i>need not</i> be authenticated to modify a document	-
5.4-3b	A role <i>must</i> be authenticated to commit the result of an activity	A user must authenticate (SSL handshake with client side certificate) to submit a container. The will also authenticate the digital signature before a container is processed.
5.4-4	A role <i>must</i> be authenticated to access production data	See 5.4-2
5.4-5	A role <i>must</i> be authenticated to access control data	See 5.4-2
5.4-6	When a role <i>must</i> be authenticated, a role <i>must</i> also be authorized	This is not a specific feature but inherent in the design. When a user has the ability to authenticate, the user will access the system with a profile that is authorized.
5.4-6a	A role <i>must</i> be authorized to commit the result of an activity	See 5.4-6
5.4-7	A role <i>must</i> be authorized to access production data	See 5.4-6
5.4-7a	A role <i>must</i> be separately authorized to access production data of separate activities	A user can only access (download) containers for workflows in which the user is listed as a current assignee.
5.4-8	A role <i>must</i> be authorized to access control data	See 5.4-6
5.4-8a	A role <i>must</i> be separately authorized to access control data of separate activities	See 5.4-7a
5.4-9	The system <i>must not</i> limit a role's organizational capacity	-

²The memory requirement itself is not a problem but combined with the run time of a transaction, it does become prohibitive.

5.4-10	Production data must only be accessible to authenticated and authorized roles	This is enforced as long as the container is in the control of the container, but once the container leaves the server, this is not longer enforced
5.4-11	Control data must only be accessible to authenticated and authorized roles	See 5.4-10
5.4-12	An activity must not be refutable	All activities are digitally signed
5.4-13	Only an activity can change control data	If an action is performed that is not according to the workflow specification, the container will be rejected by the server
5.4-14	Only an activity can change production data	See 5.4-13

Table 11.3.1 The table lists how the system design and implementation compares to the stated security objectives.

11.3.1 User Authentication

It is *only* possible to implement user authentication, when a trusted entity is able to verify the presented credentials, otherwise a malicious user may be able to influence the verification process. Hence when the provides a certificate and corresponding password to use the client application *this does not constitute authentication*, and it is not treated as such by the client application.

The client application is not usable without specifying a valid certificate and password³, but this is only to prevent unintended misuse by a user.

User authentication is performed only when a user logs on to the website and when a client application performs an XML-RPC request. This behavior also mimics the intended security objectives, as it is infeasible (or at least a separate project) to ensure that malicious activity cannot modify a container while it is outside the server's control. This means, that a (stand alone) authentication mechanism implemented in the client application cannot be trusted.

11.3.2 Enforcing Authorization

The security objectives for authorization state that a role must be authorized to access data. This definition is slightly ambiguous, as it can be interpreted as *always* or *only when system controls are in place*. In the current implementation, authorization⁴ and confidentiality are only enforced when a container is in the servers control.

This enforcement could be extended by using element level encryption in the container. Instead of storing the XML file in clear text the XML Encryption standard could be used to selectively encrypt elements, which basically means extending the schema so that for each element, it also allows an <EncryptedData> element.

11.4 System Test

The system test that has been performed is described in appendix G. The test covers unit testing of important classes, as well as a functional test of the overall system. The functional test has been limited to sequence transitions given that multi choice and multi merge are not fully functioning.

The system test has been structured to resemble the stated system requirements, hence the results of the test closely resemble the results described in the previous sections.

³Currently, the client application will also accept any valid X.509 and its password

⁴Authorization includes reader access to documents in the container, and as the container is not encrypted, anyone with access to the container can read its contents.

12

Conclusion

In this thesis I have successfully designed and implemented a prototype for a secure workflow system, that can be used to automate processes in which each decision requires formal approval by binding signature. Based on an analysis of a manual workflow, the system is designed to *be* resilient to threats a manual workflow system is *not* resilient to, and be comparable on all other identified areas.

The system automatically distributes a document container between each role in the workflow, given an immutable workflow definition stored in the container itself. The design and prototype support the simple transition patterns defined in [1] and a flexible multi choice pattern, though the complex pattern is not fully implemented in the prototype. In addition, the XML schema specification of the workflow data model, also includes inherent support for the synchronizing merge.

The container uses a versioning data model, that makes every document version available after the workflow has finished, and it is shown that this approach does not result in unmanageable file sizes. The versioning also includes a complete transaction log of how the workflow was executed, and the formal approval of each activity.

Signatures are created according to the XML Signature standard, and it is enforced that the result of an activity is always signed by the executing role, and the roles certificate identifies the role in all aspects towards the system.

The system employs an incremental trust protocol, that significantly speeds up verification of workflow execution compared to verifying workflow execution from the start node. The verification protocol also simplifies the complexity of the client, because several processing steps can be performed by the server instead.

This trust protocol also ensures that a container always contains a complete audit trail of all formal approvals given by the roles in the workflow. As the workflow definition also stored in the container, using the system effectively creates a self documenting process.

Using an XML schema to specify (parts of) the workflow model abstracts the model checking from the implementation to the XML parser, and the constraint language supported by the XML Schema Language is more suitable for this task. A validating XML parser does incur a significant increase in parser execution time, but as shown the increase is not prohibitive for actual use, even with large file sizes.

Bibliography

- [1] Will M.P. van der Aalst, Arthur H. M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 2002. (document), 4.1, 4.2, 4.2.1, 1, 4.2.2, 4.2.1, 12
 - [2] Sarbanes-Oxley. Sarbanes-Oxley Act of 2002, Public Law 107-204. <http://www.sarbanes-oxley.com/>, 1, 6.2
 - [3] Justitsministeriet. Lov om elektronisk signaturer. Retsinfo, May 2000. 1, 6.2.1, 7.3.2.1
 - [4] Justitsministeriet. Betænkning nr. 1456, e-signaturers retsvirkning. www.jm.dk, 2005. 1, 6.2.1, 7.3.2.1
 - [5] Dale Long. The Lazy Person's Guide to Workflow II. *CHIPS magazine*, October 2000. 3.1
 - [6] Open Source. Current Version Control. <http://www.cvshome.org>. 3.1.1
 - [7] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion*. O'Reilly, June 2004. 3.1.1
 - [8] Adobe Inc. Adobe Document Services. <http://www.adobe.com>, September 2005. 3.1.1
 - [9] cBrain. cBrain. <http://www.cbrain.dk>, September 2005. 3.1.1
 - [10] EMC. EMC Documentum. <http://www.documentum.com>, 2005. 3.1.1
 - [11] Microsoft Corporation. Microsoft Corporation. <http://www.microsoft.com>, September 2005. 3.1.1
 - [12] ScanJour. Scan Jour. <http://www.scanjour.dk>, September 2005. 3.1.1
 - [13] IBM. IBM Lotus Workflow. <http://www.lotus.com/workflow>, September 2005. 3.1.1
 - [14] COSA. COSA BPM. <http://eng.cosa.de/index.php>, 2005. 3.2.1
 - [15] Pallas Athena. Pallas Athena FLOWer. <http://www.pallas-athena.com>, September 2005. 3.2.1
 - [16] SAP. SAP R/3 Workflow. <http://www.sap.de/>, September 2005. 3.2.1
 - [17] Tibco. Tibco Staffware. <http://www.tibco.com/>, September 2005. 3.2.1
 - [18] FileNet. FileNet Staffware. <http://www.filenet.com/>, September 2005. 3.2.1
 - [19] Eclipse. Eclipse Platform. <http://www.eclipse.org>, November 2005. 3.2.1
 - [20] Will M.P. van der Aalst and Kees van Hee. *Workflow Management : Models, Methods, and Systems*. The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts 02142, first edition, 2004. 4.1, 4.3, 4.3.3, 7.2.2, 7.2.5.1
 - [21] Will M.P. van der Aalst, Arthur H. M. ter Hofstede, A. P. Barros, and B. Kiepuszewski. Advanced Workflow Patterns. Technical report, Component System Architecture for an Open Distributed Enterprise Management System with Configurable Workflow Support, 2000. 4.2
 - [22] Nick Russell, Will M.P. van der Aalst, Arthur H. M. ter Hofstede, and David Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. Technical report, Expressiveness Comparison and Interchange Facilitation between Business Process Execution Language, 2005. 4.2
 - [23] Stephen A. White. Process Modeling Notations and Workflow Patterns. Technical report, IBM Corporation, 2003. 4.2
 - [24] B. Kiepuszewski, Arthur H. M. ter Hofstede, and Will M.P. van der Aalst. Fundamentals of Control Flow in Workflows. Technical report, Component System Architecture for an Open Distributed Enterprise Management System with Configurable Workflow Support, 2002. 4.2.1, 4.2.1.1
 - [25] Martin Fowler. *UML Distilled*. Pearson Education Inc., 75 Arlington Street, Suite 300 Boston, MA 02116, USA, 3 edition, 2004. 4.3.2
 - [26] R. Eshuis and R. J. Wieringa. Verification Support for Workflow Design with UML Activity Graphs. In *24th Int. Conf. on Software Engineering (ICSE)*, pages 166–176, Orlando, Florida, 2002. ACM Press, New York. 4.3.2
 - [27] Alexander Knapp and Stephan Merz. Model Checking and Code Generation for UML State Machines and Collaborations. May 2002. 4.3.2
 - [28] IBM. IBM Rational Software Development Platform. <http://www-306.ibm.com/software/awdtools/>, 2005. 4.3.2
 - [29] Sparx Systems. Enterprise Architect. <http://www.sparxsystems.com>, 2005. 4.3.2
-

- [30] Popkin Software. System Architect. <http://www.popkin.com/>, 2005. 4.3.2
- [31] Institute for Software Integrated Systems. The Generic Modeling Environment. <http://www.isis.vanderbilt.edu/Projects/gme/>, 2005. 4.3.2
- [32] OMG. OCL 2.0 Specification. Technical report, Object Management Group, 2003. 4.3.3
- [33] Sundhedsstyrelsen. G-EPJ. Technical Report v2.2-20050812, Sundhedsstyrelsen, August 2005. 5.2.1
- [34] Steven Alexander. Avoiding Buffer Overflows and Related Problems. ;*login: The USENIX Magazine*, 29(1), 2004. 3
- [35] Peter Baer Galvin. Solaris 10 Containers. ;*login: The USENIX Magazine*, 30(5), 2005. 3
- [36] Josef Pieprzyk, Thomas Hardjono, and Jennifer Seberry. *Fundamentals of Computer Security*. Springer-Verlag Berlin Heidelberg, Heidelberg Platz 3, 14197 Berlin, Germany, 1 edition, 2003. 5.3, 6.1.1, 2, 6.1.4.1
- [37] Revisionsteknisk Udvalg. RS314. <http://www.fsr.dk>, December 2004. 5.3
- [38] Revisionsteknisk Udvalg. RS315. <http://www.fsr.dk>, December 2004. 5.3
- [39] Charles P. Pfleeger. *Security In Computing*. Prentice-Hall Inc., Upper Saddle River, New Jersey 07458, 2nd edition, February 2000. ISBN 0-13-337486-6. 5.3, 5.3.1, 6.1.1, 6.1.4.1
- [40] William Stallings. *Cryptography and Network Security: principles and practices*. The William Stallings Books on Computers and Data Communications Technology. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 2nd edition, 1999. 5.3.1, 6.1.4, 6.1.4.1, 6.3.1.1, 7.3
- [41] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1 edition, 1996. 6.1.1, 6.1.2, 6.1.2, 6.1.3, 6.1.4.1
- [42] Health Insurance Portability and Accountability Act of 1996. Library of Congress, August 1996. (Public Law 104-191 104th Congress). 6.2
- [43] IT- og Telestyrelsen. OCES Certificate Policies. www.signatursekretariatet.dk, 8, 6.3.1.1
- [44] Mads Bryde Andersen. IT-retten. <http://www.itretten.dk/>, September 2005. 6.2.1
- [45] IT-Sikkerhedsrådet. Digitale dokumenters bevisværdi. Technical report, IT-Sikkerhedsrådet, December 1998. 6.2.1
- [46] Stephen Kent and Tim Polk. PKIX Working Group. <http://www.ietf.org/html.charters/pkix-charter.html>. 11
- [47] B. P. Aalberts and S. van der Hof. Digital Signature Blindness - Analysis of legislative approaches toward electronic authentication. November 1999. 6.3
- [48] ObjectWeb. Enhydra JaWE: Java Workflow Editor. <http://jawe.objectweb.org/>, September 2005. 7.2.5
- [49] Gocept. Grafical Workflow Editor for AlphaFlo. <http://www.gocept.com/>. 7.2.5
- [50] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. The Role of Trust Management in Distributed Systems Security. In *Secure Internet Programming*, pages 185–210, 1999. 7.3.1
- [51] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures (Position Paper). *Lecture Notes in Computer Science*, 1550:59–63, 1999. 7.3.1
- [52] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. *Proceedings IEEE Conference on Security and Privacy*, (96-17):10, May 1996. 7.3.1
- [53] Brandon Palmer and Jose Nazario. *Secure Architectures with OpenBSD*. Addison-Wesley, 75 Arlington Street, Suite 3000, Boston, MA 02116, 1 edition, 2004. 7.3.1
- [54] Massachusetts Government - Information Technology Division. Enterprise Technical Reference Model - Version 3.5. <http://www.mass.gov/>, December 2005. 7.4.1
- [55] Valoris. Comparative Assessment of Open Documents Formats Market Overview. Technical report, European Commission, December 2003. 7.4.1
- [56] Telematics between Administrations Committee. TAC approval on conclusions and recommendations on open document formats. <http://europa.eu.int/idabc/en/document/2592/5588>, May 2004. 7.4.1
- [57] Erik T. Ray. *Learning XML*. O'Reilly, 1005 Gravenstein Highway North, Sebastopol, CA, 2 edition, 2003. 8.2
- [58] OpenOCES. www.openoces.org, December 2005. 10.3.1
- [59] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of reusable object-oriented software*. Addison-Wesley, Pearson Education Corporation Sales Division 201 W. 103rd Street Indianapolis, IN 46290, 2004. 3
- [60] Jothy Rosenberg and David Remy. *Securing Web Services with WS-Security*. Sams Publishing, Sams Publishing, 800 East 96th Street, Indianapolis, Indiana 46240, 2004. 10.5



Contents on CD-ROM

The supplied CD-ROM contains the following.

source_code

The `source_code` directory contains all sources used to develop the prototype. The prototype has been developed using the Eclipse development tool, and the directory is simply a container for the corresponding Eclipse project.

Apart from the `net.strandbygaard.xflow.*` source tree the project also contains a source tree with the sources of Apache XML Security library (that is used to perform all XML signature and encryption operations), and the project has been compiled against this source tree.

The source tree was included to make it easier to debug the XML signing code.

documentation

The `documentation` directory contains three sub directories. The `javadoc` directory holds the Javadoc documentation generated using the SUN Javadoc tool. The `doxygen` directory holds documentation created using Doxygen, and the `schema_documentation` contains documentation generated from the inline documentation in the actual schema file. The schema documentation includes diagrams of all major constructs in the schema.

The SUN version of the Javadoc is more suited as a reference during development, whereas the Doxygen version makes it easier to obtain an understanding of how the system works.

www

The `www` directory holds all files comprising the web page, that users access to download container files.

report

The `report` directory contains this report in PS and PDF formats. For online viewing, the PDF version is recommended, as all links, cross references, and index can be used to navigate the document.

B

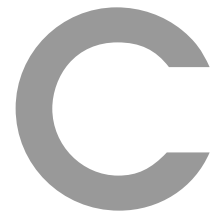
Comparison of Security in Manual and Electronic Workflow Systems

This appendix contains an evaluation of a “manual” or “paper based” workflow system. The evaluation assumes a typical office environment, in which internal (paper) mail is distributed using the ubiquitous brown “internal mail” envelope, which is sent by placing it in the “outgoing” pigeon hole, and received by being delivered to one’s own pigeon hole.

Table B.0.1: Security in Manual Workflow System	
Threat Macro	Description
5.3-1	Very resilient. Mallory is an external threat, and it must be assumed that physical security is not a problem
5.3-2	Not resilient. In a typical office environment, paper is left on desks or in unlocked drawers, hence getting access to other people’s papers is not a problem
5.3-3	Very resilient. As in 5.3-1 Mallory is external hence she cannot send any documents internally, because she cannot get inside.
5.3-4	Depends on internal procedures. If a document is submitted simply by putting it in an “internal mail” envelope and having sent it to a different department, then it is not a problem, but if the document must be signed, it becomes more difficult.
5.3-5	Very resilient. See 5.3-1
5.3-6	Not resilient. See 5.3-2
5.3-7	Not resilient. Alice can easily intercept Bob’s internal mail, and resend it to Carol
5.3-8	Very resilient. See 5.3-1
5.3-9	That depends on what is being sent. If it is a printout, then creating a duplicate is not a problem, but if it is an original invoice, creating a duplicate is not straight forward.
5.3-10	Depends. See 5.3-9
5.3-11	Very resilient. A manual workflow system does not become unavailable, except if the mail clerk is out sick, and even then a substitute can easily be found.
5.3-12	Very resilient. See 5.3-11
5.3-13	Very resilient. See 5.3-11
5.3-14	Very resilient. See 5.3-11
5.3-15	Very resilient. See 5.3-11
5.3-16	Not resilient. See 5.3-2

5.3-17	Not resilient. See 5.3-2
5.3-18	Not resilient. See 5.3-2
5.3-19	Very resilient. See 5.3-1
5.3-20	Very resilient. See 5.3-1
5.3-21	Depends. See 5.3-9
5.3-22	Very resilient. See 5.3-1
5.3-23	Depends. See 5.3-9

Table B.0.1 The table lists how secure a manual system is in terms of the stated threat macros.



Use Case Diagrams

User - Get Container

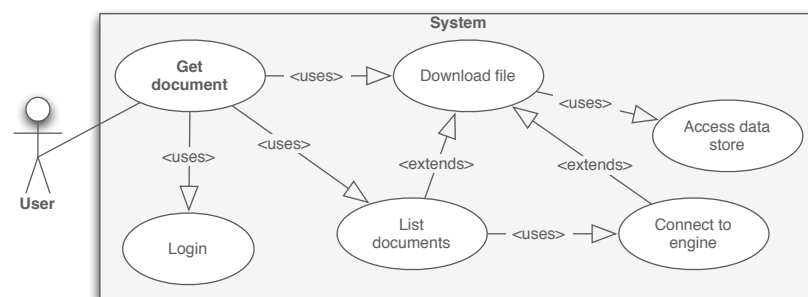


Figure C.0.1 The complete use case for retrieving a document from the system.

User - Submit Container

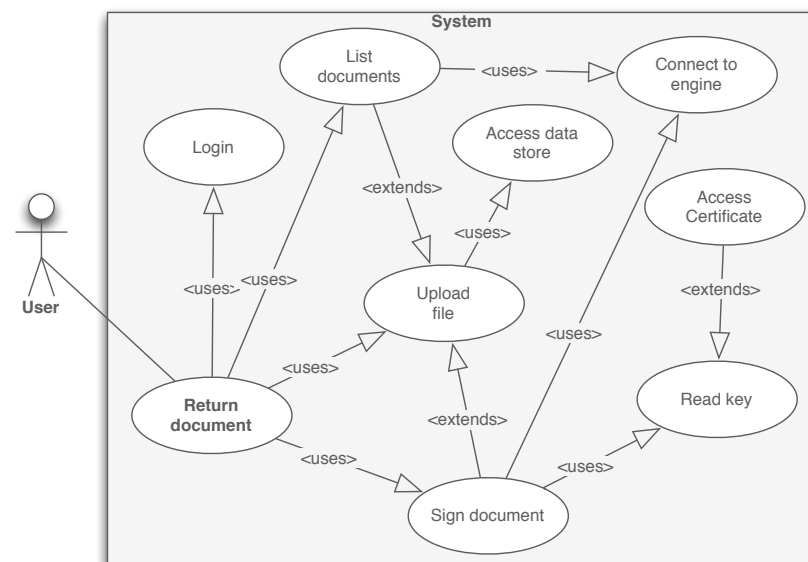


Figure C.0.2 The complete use case for submitting a document from the system.



The Client User Interface

This appendix describes the user interface of the client application, and gives a detailed walk-through of what the individual components are used for.

As such this chapter also serves as a “user guide” for the client application. The first section introduces the widget toolkit used in the GUI, and the following sections describes the individual GUI components.

D.1 Overview

The user interface is implemented using the freely available widget toolkit Thinlet (www.thinlet.com), that has a number of advantages compared to the standard Java Swing library. Besides being very light weight (jar file is 40kb), its main advantage is that it allows the entire GUI to be described as an XML document, that is parsed at run-time and used to render the application window.

Specifying a GUI as an external entity instead of writing code to generate it, makes for a much cleaner implementation of the Viewer part of an MCV model, but it does have the disadvantage, that all callback handlers on GUI components must be implemented in the same class (only true for statically typed GUI objects). This means that a GUI callback handler class can grow very large, but with the limited complexity of the current client, this is not a problem.

D.2 Screen Elements

The GUI is divided into two parts: The upper part does not change and contains functionality that is always required to use the client, e.g. specifying a certificate. The lower part of the application window is a tabbed area, in which each tab represents an isolated area of functionality. The GUI is designed to enable and disable GUI components automatically, as the application state changes, e.g. a user will be unable to submit a container before it has been signed.

D.2.1 Common Objects

The upper 30% of the application window contains the static or common GUI objects that do not change, and which are always required. This part is shown in the following figure:

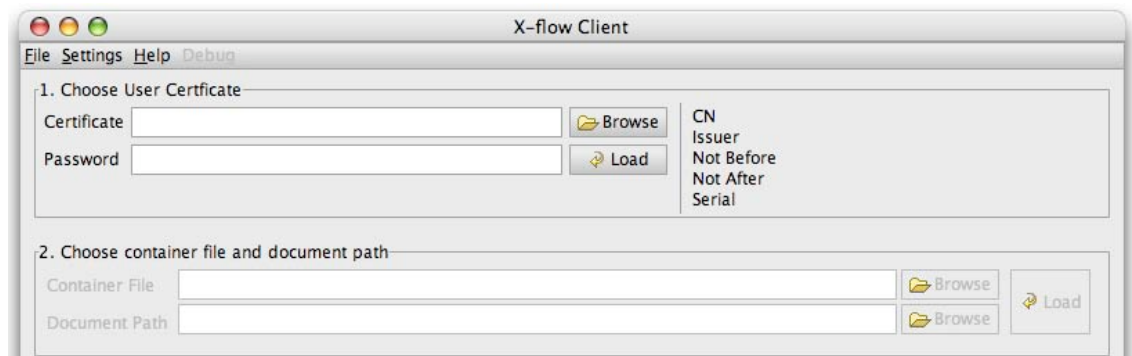


Figure D.2.1 Common fields in the GUI

D.2.1.1 Certificate Section

The `certificate` path is used to point to the certificate identifying the current user and must currently be in the form of a PKCS12 file. The `password` field is used to enter the password protecting the private key stored in the PKCS12.

Before the certificate is available to the client program, it must be *loaded*, which is done by pressing `load`, upon which an initial program state is created. To the left of these fields, an information area will print out key information about the specified certificate, after it has been loaded.

When a certificate is loaded, the `Certificate` and `Password` fields will be disabled, and the `Load` button will be changed to an `Unload` button, which will do just that, and re-enable the fields.

D.2.1.2 Container Section

When a certificate is loaded, the fields `Container File` and `Document Path` will be enabled. The former is used specify a path to the container file, that should be loaded, and the latter should specify a folder, where the document in the container should be stored. The client program will automatically store the newest version of a document to the specified path, once the container is loaded (overwriting any existing version!). When both have been specified, the container is loaded using `Load`.

Similar to the certificate section, the container section will also be disabled when a container is loaded, and the `Load` button will be changed into an `Unload` button.

D.2.1.3 Menu Bar

Currently, the menu bar can only perform a few options, but including it in the design makes it easier to extend the GUI specification at a later stage. In the file section it is possible to open a container and exit the program, and in the settings section is shown a number of example certificate devices.

D.2.1.4 Status Bar

The lower part of the GUI is a status bar, that is used to print status messages to the user. The status bar is linked to the internal event model of the program as well as the enabled log levels, meaning that the status bar will print different messages depending on the log level that is enabled.

D.2.2 Info Tab

The info tab is used to view state information about the currently open container, including information about the server that is being used as broker in the workflow, and the certificate presented by this server. This tab also includes information about the current step, and the user that signed the document in the previous step (currently this does not handle parallel step).

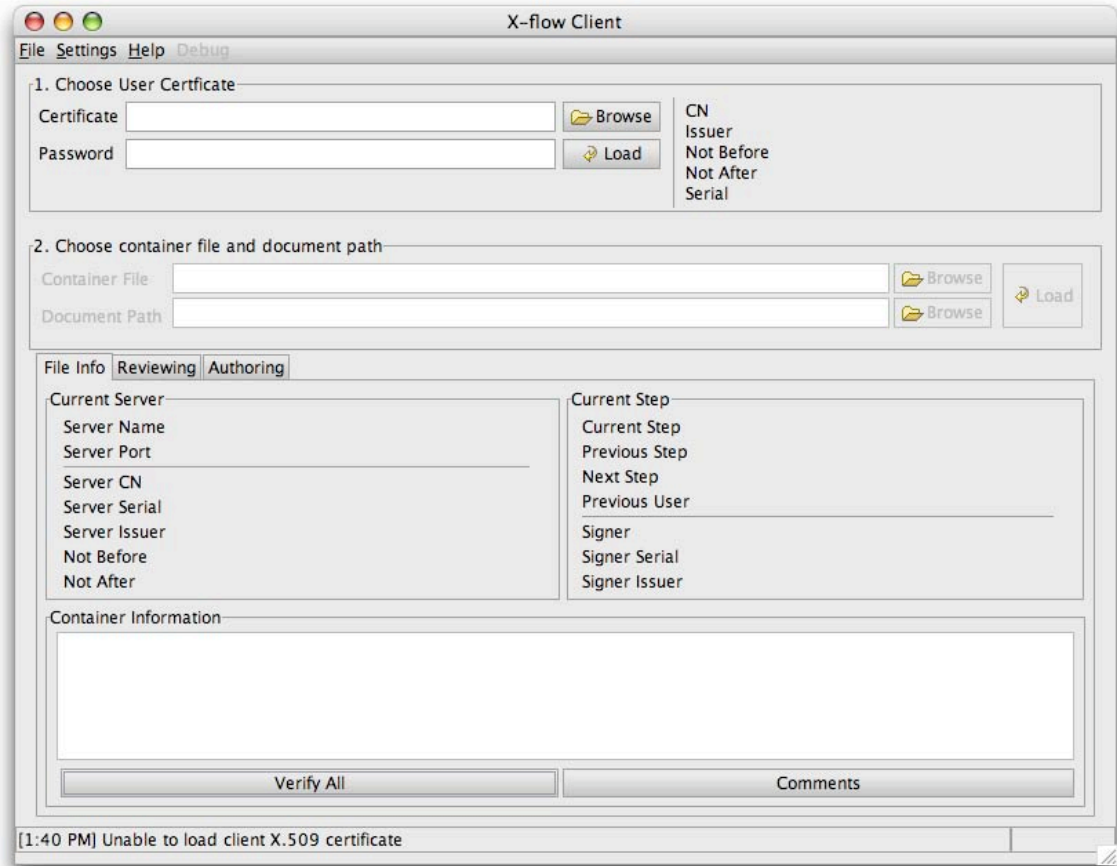


Figure D.2.2 Tab to access to information about the loaded container. Here the tab is shown together with the top part of the GUI.

The client program bases its trust model on the receipt in a received container, however, if for some reason, a user wish to independently verify the validity of a container, this can be done using the `verify` button.

Finally, a user may extract a list of all comments in the container. The client does not implement any element encryption which means that any user will be able to view all comments.

D.2.3 Reviewing Tab

The reviewing tab is used, when a user should review a container. The container specification includes a number of extended reviewing facilities, which have not been implemented. The reviewing tab includes the basic functionality required to review a document, which includes assigning it one of 3 fundamental states (approve, reject, finalize), and optionally providing a comment (and comment file).

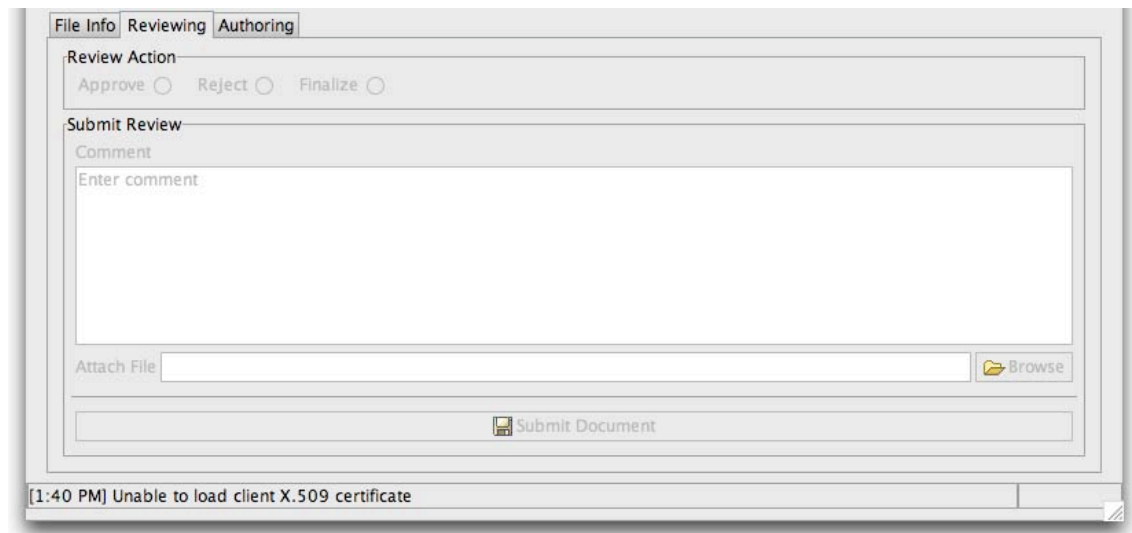


Figure D.2.3 Tab used when reviewing a document

The radio buttons in the top part control which review action, or status, should be assigned the document in the container, and the lower part allows the reviewer to enter a comment, and attach a file to the container.

D.2.4 Authoring Tab

The final tab is used when the document stored in the container should be modified and updated to a new version.

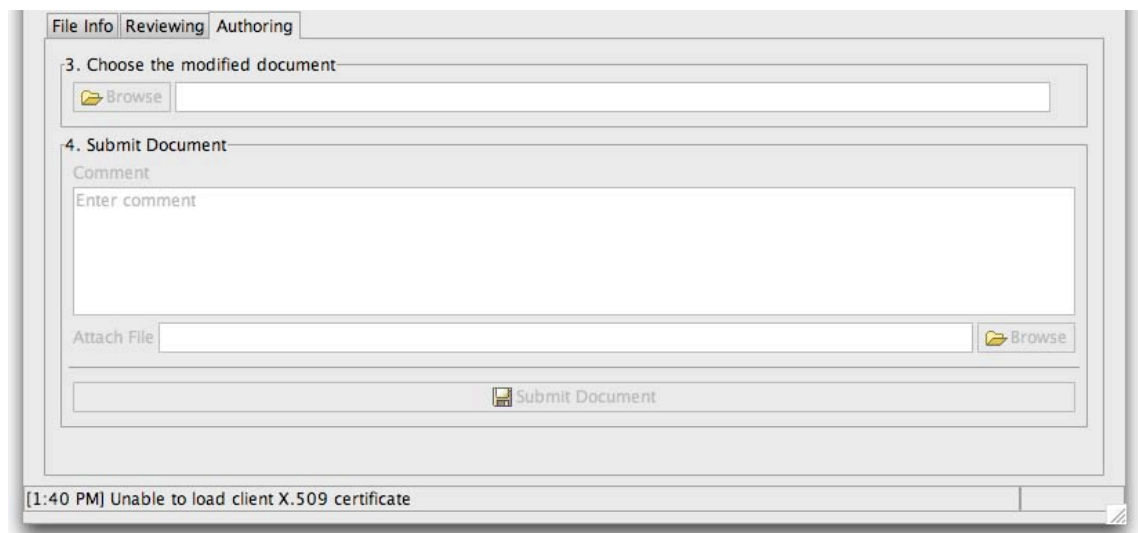


Figure D.2.4 Tab used when modifying a document

The author tab is almost identical to the review tab except the review actions have been replaced with a field in which to enter the path to the modified document, that should be added to the container. Besides the modified document, an author also has the option of attaching a comment and a comment file.



The Web Interface

This appendix describes the implementation and organization of the supporting website, and the functionality that is available to the user and the administrator. A picture of each page, that is part of the user interface is shown along with a brief description of the functionality available through that page.

The first section describes the design and implementation of the website, the second section describes common pages and user pages, and section three describes the pages used by the administrator.

E.1 Overview

The website provides download access to currently waiting containers, and an administrator can create and delete users and containers.

The website is implemented in PHP v4 as a separate tier between the SQL database in which the server stores containers awaiting user action, and the client. The website was developed and tested using the Apache (v1.3) webserver with PHP module, but should work with any webserver that supports PHP v4.

The client-server system is backed by an SQL database, and the database schema is extended with fields used by the web page. Specifically, the client-server system only uses X.509 certificates, and to support clients that don't support authentication with live connect and HTTPS an additional username/password authentication option has been added to the website. This only has limited affect on the overall security model, as the container itself is responsible for enforcing access control.

It should be noted that the username/password authentication method implemented here is solely for accessing the website, and is not used anywhere else in the system.

E.1.1 Standards Compliance

The website design uses XHTML and CSS as client side presentation technology, and all code sent to a client will validate according to the relevant W3C standards.

Given the current state of web technologies, and their market adoption, designing the presentation layer of a website is always a compromise between:

1. Functionality
 2. Ease of implementation
 3. Client support
-

A web design that uses only valid XHTML and CSS will not be rendered correctly by most browser implementations, as *all* common browsers have known bugs in their support of these two technologies. However, these are the core standards promoted by W3C, and consequently the ones that should be used.

The website has been tested successfully in Internet Explorer v6, Mozilla Firefox, and Apple Safari, all of which were able to show the pages correctly, and given the simple design of the site most all browsers that support XHTML and CSS should be able to view the site.

E.1.2 Security

Creating a *secure* website that is used for much of anything else than serving public static content, is a project in itself, and has not been the focus of this thesis. Avoiding typical security problems in website design takes a great deal of planning, and a great deal of code to handle parameter validation, content transformation and more.

This website is intended to demonstrate the features that are required from the system, and as pointed out in the report many functions are best handled in other systems, especially user- and role administration. Consequently, the design of this site is vulnerable to a number of common security problems in web applications, specifically:

- SQL injection
- Client side scripting
- HTML input validation
- HTTP parameter validation

A *user* will be able to execute arbitrary SQL code with the same permissions as the PHP user accessing the database, because input from a user is used, non-validated, in SQL queries. If a user enters e.g. `' ; DELETE FROM users where username=*` as username in the login box, that would delete all users in the database¹.

Because PHP is not a strongly typed language solving the problem using e.g. prepared statements as is done in Java is not possible, however, the data can be checked using e.g. a regular expression to see if it contains disallowed patterns.

An *administrator* will be able to have a script executed in the browser of a connecting user, by inserting a (Javascript) script into a text field that is shown to the user, and if the website is extended to allow users to enter comments, a user will be able to do this as well.

The two common approaches to solving this problem is using a character blacklist and HTML encoding all data sent to a client. HTML encoding data sent to a client solves the problem on the client side (e.g. no script will be executed, if it is HTML encoded), but the user can also send a script, that will be stored in the database, and storing data HTML encoded is not a good solution (as it links content and presentation). A blacklist solves this problem by disallowing characters that are generally used in scripts, while at the same time rarely used in regular text (e.g. `<`, `>`, `#`, `$` and others).

HTTP parameter validation is an issue, because not all parameters passed as HTTP parameters are validated, meaning it is not ensured that a user is authorized to access data that is requested. The most obvious example is probably when a user downloads a container, where the container to download is determined by the `id` passed to the page `download.php`. If it is not verified that the user actually has access to the requested container, an *authenticated* user will then be able to download *any* container in the system.

¹Dropping all tables is usually not a *good* approach as the current database normally doesn't have those permissions. However, users can be deleted from the administrative interface, so the database user must have those permissions.

This is one of the most common errors in website design and typically the cause, when it is publicized *that all users can see all other user profiles in website XYZ*. It is validated that a user is authorized to retrieve a certain container, by appending the active users DN (stored in the user's session), as a conditional on the SQL query that returns the container². However, similar validation is not performed on all admin pages, where it is only tested that the current user is an administrator, hence HTTP parameter validation may be an issue for the administrative interface, if the administrator user/role model is expanded to include different levels of administrators.

The impact of these issues can be mitigated if users are only allowed to login using X.509 certificates, and if `id` values passed to `download.php` are validated to ensure they only contain numbers, as that would remove the possibility of SQL injection, which is the most problematic of the 4 issues.

E.1.3 External Configuration

The security of the website depends on the configuration of the webserver on which it is deployed and HTTPS is configured on the webserver itself. The webserver should be configured, so that HTTPS is required for all pages except `index.php`, and `login.php` should attempt to request a client certificate.

Ideally the webserver should require client certificate to establish an HTTPS connection, but obviously this will stop username based authentication.

²By extension of the discussion concerning SQL injection, a user will also be able to perform SQL injection here.

E.2 User Interface

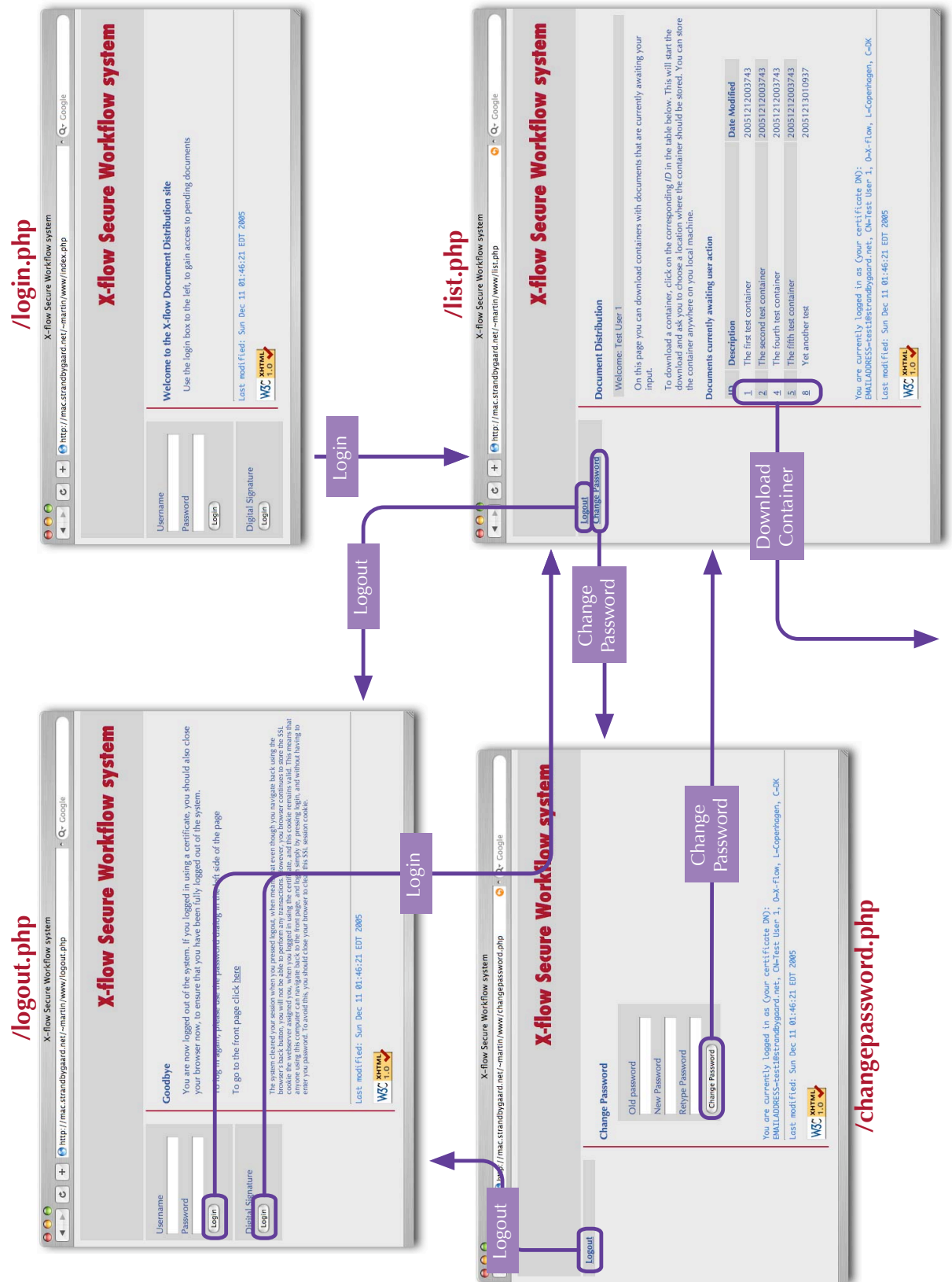


Figure E.2.1 The figure shows the sitemap, and page interaction of the pages that are used by a normal user of the system. A user has the option of logging in, logging out, changing password, and viewing/downloading containers.

The website contains three pages that are common to a regular user and an administrator: login, logout, and change password.

E.2.1 /login.php

It is possible to login from the main page, using the username/password dialog in the left side of the screen, or using the certificate option below. Before a user (or an administrator) can login, a corresponding user account must be created in the `users` table.

A user is logged in using username and password by comparing the password supplied by the user, to that stored in the database (of the specified user). If a user logs in using a certificate, the webserver forces an SSL context, and the client DN in this context is then looked up in the `user` table. In either case, if a match is found, the user is logged in (by creating a PHP session, and setting a variable in this session).

The login dialog is presented by `index.php`, and the actual login is performed by `login.php`

E.2.2 /changepassword.php

The change password dialog allows the user to change the current password to a new one. The dialog does require the user to type the new password twice, and the two entries must match, but the system does not enforce any complexity requirements, and even the empty password may be chosen.

The change password dialog is presented and performed by `changepassword.php`.

E.2.3 /logout.php

The logout function is available as a left side menu selection in all pages of the website. When a user logs out, all variables stored in the user's session is unset, and the session is destroyed, however a user may still access previously visited pages using the back-button.

The logout function is performed by `logout.php`

E.2.4 /list.php

From this page the user is able to download waiting containers. A container is downloaded by clicking the appropriate link and when a user submits a modified container, the old one is removed from the user's current listing.

The list of waiting containers is presented by `list.php`, and a container is downloaded by calling `lib/download.php` with the desired container as parameter.

E.3 Administrator Interface

E.3.1 Managing Users

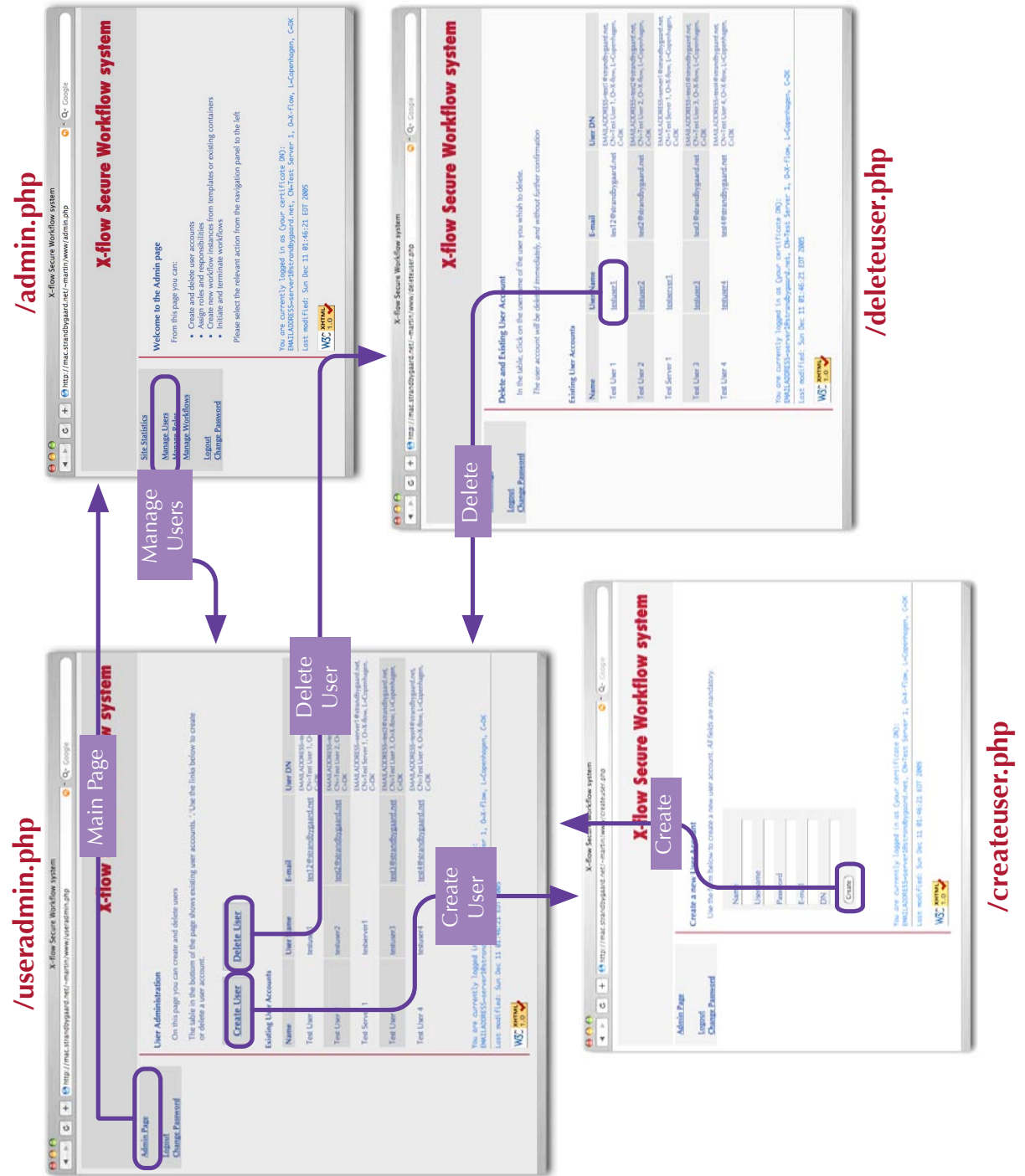


Figure E.3.1 The figure shows the sitemap, and page interaction of the pages that are used by an administrator to manage users in the system. From these pages an administrator can view, create, and delete user accounts in the system.

E.3.1.1 /useradmin.php

This is the main page for all user administration. From this page, an administrator can see all created users, and there are options to create and delete a user account.

E.3.1.2 /createuser.php

The dialog used by an administrator to create a user account. The dialog contains 5 fields, and while it is not tested, all fields should be filled out, as they are all required by the system.

When the administrator presses create, a user account is created in the database, and `useradmin.php` is shown again.

E.3.1.3 /deleteuser.php

The dialog used by an administrator to delete a user account. The administrator is presented by a list similar to the one shown by `useradmin.php`, but in this list, the username field is now a link, and clicking on a username, will delete the corresponding user in the database. Now warning box is presented before the user is deleted.

When the user is deleted `useradmin.php` is shown again.

E.3.2 Managing Roles

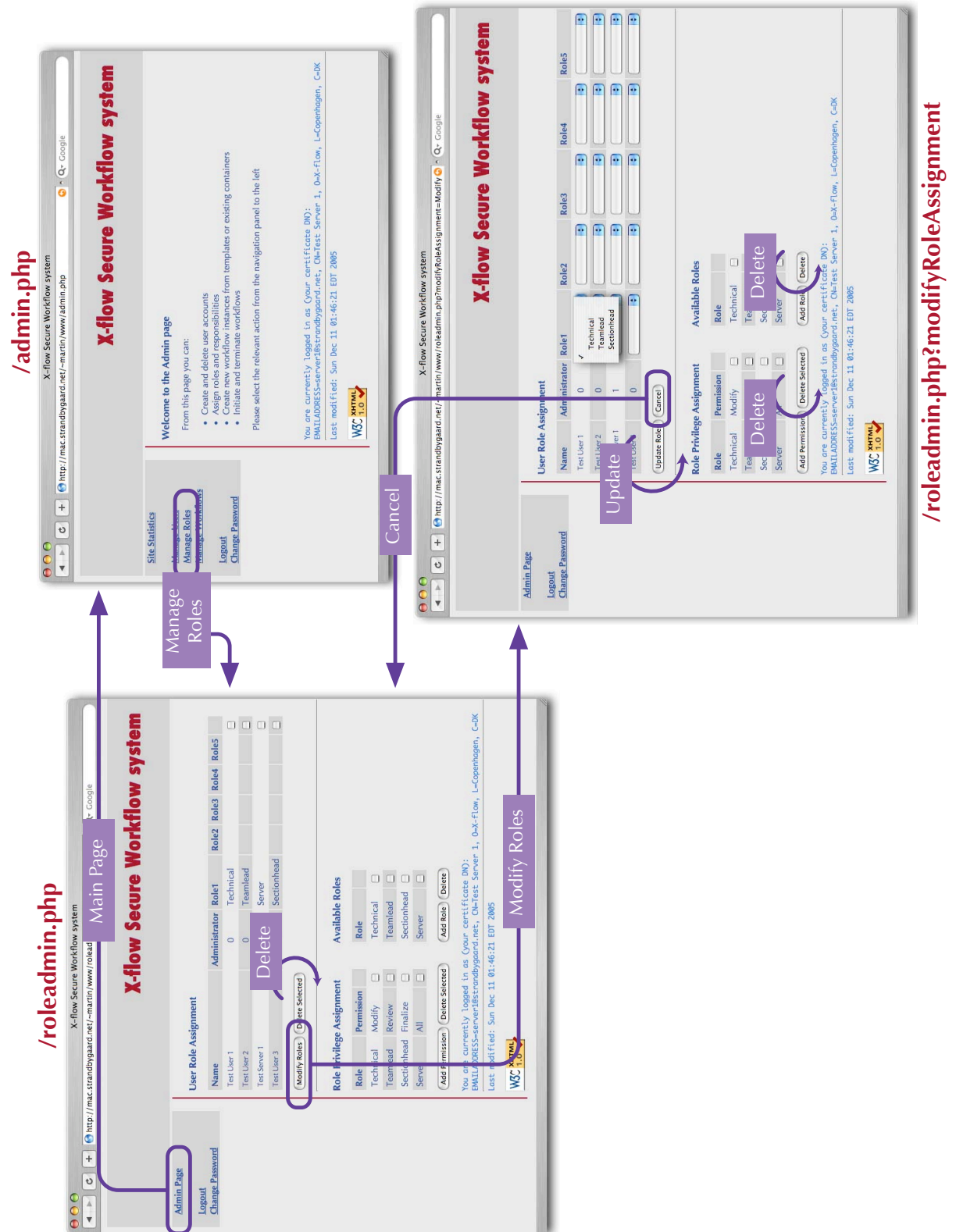


Figure E.3.2 The figure shows the sitemap, and page interaction of the pages that are used to manage roles in the system. When selecting *Role Management* a page is presented, that shows an overview of permissions, roles, and assignments. From this page it is possible to modify each part.

E.3.2.1 /roleadmin.php

An administrator uses this page to change all aspects of roles and their assignment, including creating and deleting roles, and changing user role assignment.

The page is implemented with two *states*, one representing the overview page, and the other the modify page.

E.3.3 Managing Workflows

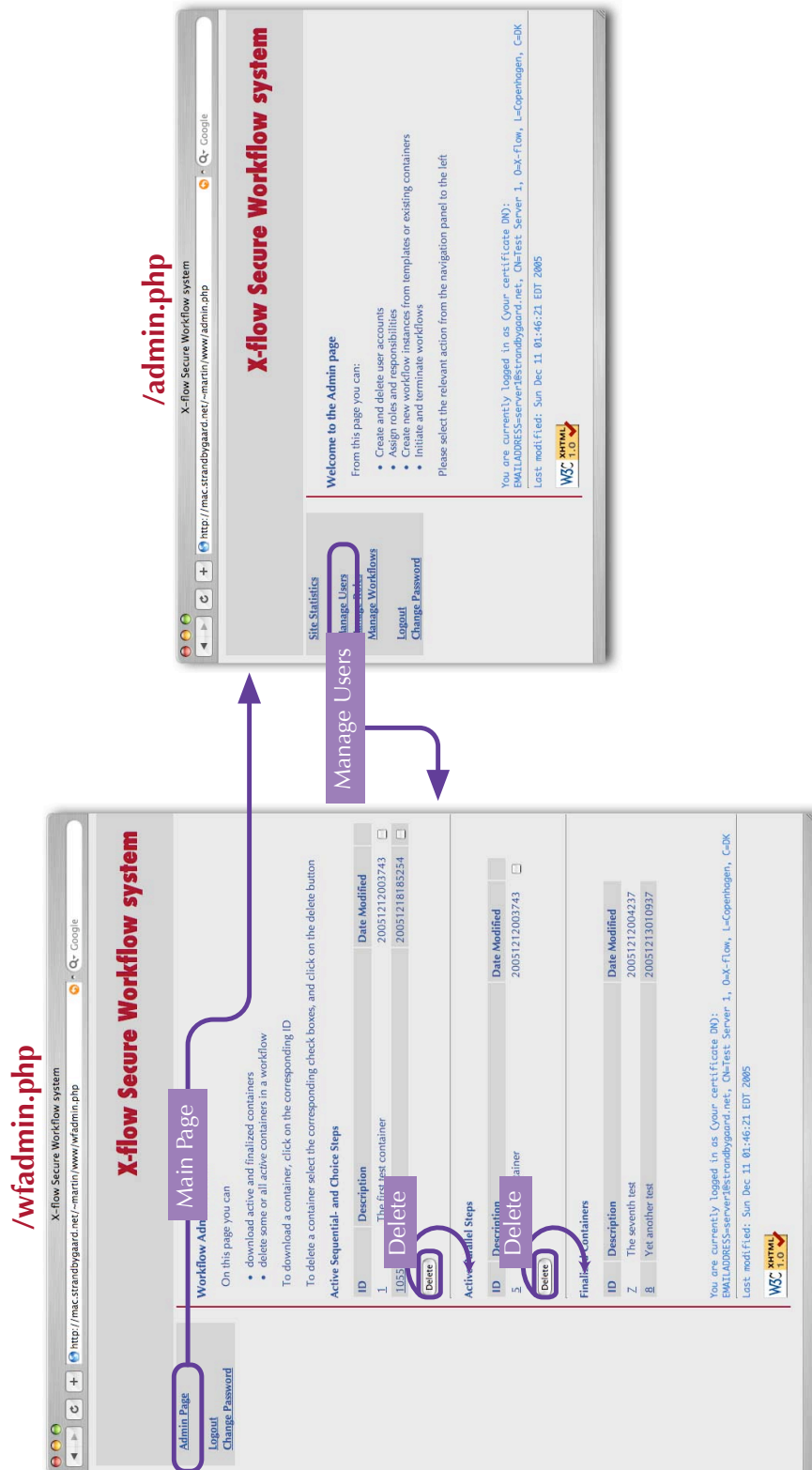


Figure E.3.3 The figure shows the sitemap, and page interaction of the pages used by an administrator to manage currently active workflows. An administrator has the option of downloading all containers in the system, and the administrator can delete active containers.

E.3.3.1 /wadmin.php

From this page, an administrator can delete active containers. Once a container has been finalized, it is locked in the system, and the admin can no longer delete it.



Admin Tool

The command line tool `XfAdminTool.java` can be used to perform a number of actions, that are necessary to “bootstrap” a workflow, such as getting a container “into” the workflow. The full usage of the command line tool is shown below:

```
usage: XflowClient
-certificate      PFX file containing certificate and private key
-clean           Clean instance data
-cleanall        Clean instance data and documents
-comment         Provide a comment
-container       Container file
-file            Modified file to be submitted
-getCommentDocument Get the specified comment file
-getComments     Get a list of all comments in the container
-getDocument     Get the document stored in the specified element
-init           Convenience method to clean and sign (info,
               workflow, 1st receipt) a template
-password       Password for PFX file
-receipt        Generate a new receipt
-sign           Sign an element in a container (enveloped)
-submit        Submit a completed container
-test          Start a test server, and use this (must be
               specified in the container)
-validate       Validate the specified container
```

The tool is not very intelligent in its argument handling, hence only one function should be attempted at the same time (e.g. don't call it with both `-sign` and `-receipt`).

Invoking the tool with a function that works on a container, requires that the container path is specified using the parameter `-container`, and additionally if signing must be performed (`-sign`, `-receipt`), a certificate and password must also be specified using `-certificate` and `-password`.

The `-comment` parameter, and all `-get*` parameters have not been implemented, but the client program will fill in these gaps.



System Test

System test is divided into two parts:

- Unit testing that is used to perform structural testing, and which is developed simultaneously with the application code.
- Functional testing that tests the intended functionality of the application, and which is performed on the finished system, possibly using dedicated scripts or programs

As part of completing the *X-flow* system, both unit- and functional testing has been performed, and the following sections describe the scope of each test phase.

Unit testing has only been performed on parts of the Java code.

Because the XOR-split and synchronizing merge patterns have not been fully implemented, it is not possible to test a workflow with branches.

G.1 Test Data

The system has been tested using two different workflow specifications, that are representative of two common use cases:

- Sequential modification of a document (sequential process)
- Sequential modification with parallel review and final approval (parallel review)

G.1.1 Sequential Process

This is probably the most common of all organizational processes, and exist whenever one person writes a document that others must either change or review. The following use case is assumed:

As part of a project an employee prepares a draft report, which he sends to the person heading the project, for his/her view and feedback. Upon receiving this feedback, the employee then incorporates this information into the draft document, removes the draft status, and forwards it to the project leader for publishing as part of the official project documentation.

Expressed formally as a UML state diagram, this process can be expressed as:

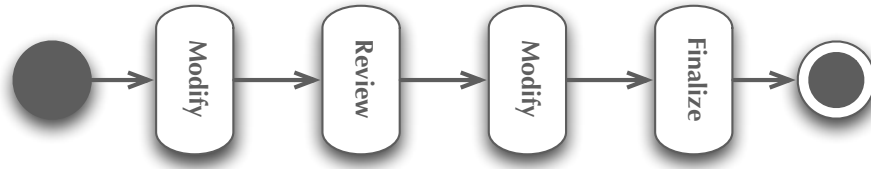


Figure G.1.1 UML diagram of sequential process

The label of each state denotes the action that takes place in this state. This process description is then mapped to the following system model (graphical representation of the container instance):

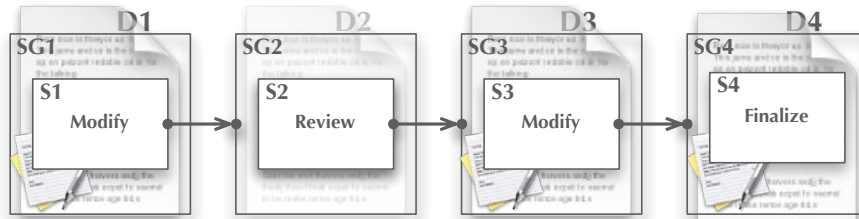


Figure G.1.2 System internal model of workflow in G.1.1

Each UML state is mapped to a `<xf:stepGroup>` element, with the step group type (`<sequence>` or `<parallel>`) determining the number of possible steps ($n = 1$ or $n > 1$) in the group. In this case, only one step occurs in each group.

The representation in G.1.1 also shows the values of the `<xml:id>` references pointing to each element in each state.

G.1.2 Unit Testing

The unit testing is focused on member functions that access or update data elements, e.g. the value of an XML element. Most class methods can be categorized as either:

1. modifying the overall class model, or
2. performing low level operations on the in-memory `org.w3c.dom.Document` instance of the current container file.

The unit tests accompanying the *X-flow* system only focus on the methods in 2, and test of higher level methods is primarily addressed by the functional testing.

The methods that perform high level operations are less well suited for unit testing as it becomes much more difficult to setup a well defined test case, and because the actual unit (of code) under test isn't very well defined. E.g. if a high level method calls a number of other methods it is difficult to determine which code unit is actually being tested.

To some extent functional tests will also test higher level methods, as a number of these closely represent the intended functionality.

G.1.2.1 Test Setup

Unit testing is performed using the Junit unit testing library, and the unit tests have been developed according to established practices for using Junit, which implies:

- There is no cross section between test- and production code
- The code is divided into separate source trees (and packages)

- Each unit under test is named as `Test<item>`, e.g. the unit test corresponding to the source file `Workflow.java` becomes `TestWorkflow.java`, and a *test*-method to test a member in `Workflow.java` is named `TestMember`

G.2 Test Results

The following sections present the result of the system case.

G.2.1 Unit Test

Unit tests have been implemented for the following classes (8 of 83 classes).

- `XflowTrustAPI.java`
- `SimplePKCS12File.java`
- `PKCS12File.java`
- `Step.java`
- `Container.java`
- `ClientContainer.java`
- `ClientWorkflowEngine.java`

Time did not permit implementing unit tests for all classes, hence the classes that comprise the major part of the primary code path (of the client) was chosen.

G.2.2 Functional Test

This section presents the results of the functional test, which are summarized in table G.2.1.

The table contains a test reference, a description of each functional test case, the result of the test, and a description of the test result, if the test failed.

The result is represented by a three-color code (`green`, `yellow`, and `red`), where `green` means that a test completed without problems, `yellow` means that the test completed but only with limited success, and `red` means that the test was not successful.

The reference links the test case to a labeled part of the requirements specification, design, or implementation, covered in one of the earlier chapters of this report.

Table G.2.1: Functional Test		
Test Description	Comment	Ok
Web Access (User)		
Log in to the web site using username and password		green
Log in to the web site using certificate		green
Change current user password		green
Log out of the website (username/password)		green
Log out of the website (certificate)	User must close browser before actually being logged out. This is common to most all web applications that use client-side certificates	yellow
View waiting containers		green
Download waiting container		green
Web Access (Administrator)		

Log in to the web site using username and password		
Log in to the web site using certificate		
Change current user password		
Log out of the website (username/password)		
Log out of the website (certificate)	User must close browser before actually being logged out. This is common to most all web applications that use client-side certificates.	
View Site Statistics	Not implemented	
User Management		
Create a new user		
Modify an existing user	Not implemented	
Delete an existing user		
Role Management		
View assigned roles		
Modify assigned roles		
Add user-role assignment	Not implemented	
Assign new role	Limited to 5 roles	
View role privileges		
Assign role privilege	This feature is available in the back-end, but has not been shown on the page.	
Delete role privilege		
View available roles		
Add role	Not implemented	
Delete role		
Workflow Management		
View list of active containers in sequential steps		
Download active container in sequential steps		
Delete active container in sequential steps		
Reassign active container in sequential step	Not implemented	
View list of active containers in parallel steps		
Download active container in parallel steps		
Delete active container in parallel steps		
Reassign active container in parallel step	Not implemented	
View list of finalized containers		
Download finalized container		
View information about a container	Indirectly supported, by downloading the container and viewing it with an XML editor or the client tool.	
Client Application (XfClient)		
Load certificate		
Load container		
View information about current receipt		
View information about the previous user	Is part of the implemented event model, but is currently not extracted from the loaded container.	
View comments in container		
Extract comment files from container	Not implemented	
Extract the current document from the container		
Add a modified document		
Add a comment		
Add a comment document		
Sign a container		

Submit a modified container		
Submit a reviewed container		
Access meta data element	No meta data support is implemented (dublin core and controlled vocabulary from the schema spec.). Regular comments and comment files are supported.	
Workflow Server (Xflowd)		
Validate received container	The system is unable to validate a container that contains joined steps (multiple submissions combined into one container). A joined step will always validate	
Select next step group in a sequence		
Select next step group in <i>after</i> a parallel step		
Select next step group <i>in</i> a parallel step	The system is unable to join multiple submissions in a parallel process into one document. The system also does not know how to determine if the parallel process has ended.	
Conditionally select next step group	Expression evaluation is implemented as part of <code>ProcessorThread.java</code> , but <code>Selector.java</code> has not been implemented, which means that the first expression always evaluates to <code>true</code>	
CLI administration tool (XfAdminTool)		
Sign an element in a container		
Generate new initial receipt on container template		
Submit an initialized container into the workflow	Implementation is not very robust, and will only handle certain cases.	
Submit a container as a client		
List all comments in a container		
Retrieve document from container		
Retrieve comment document from container	Not implemented	
Validate a container	Will only validate signatures within the container. No other validation is performed.	

Table G.2.1 Results of the functional test.



Source Code

The *X-flow* system is developed on Unix using the Eclipse development environment (the Eclipse project is provided on the supplied CD).

The system is specifically designed for Java version 1.5, as it makes use of type safe lists and the updated iterator interface, both new in Java version 1.5. This also means that the system will only compile and execute on a Java version 1.5 platform.

The source code is organized in the following namespaces:

net.strandbygaard.xflow.client(*) Implements functionality that is only used by the client program

net.strandbygaard.xflow.server Implements functionality that is only used by the server

net.strandbygaard.xflow.container Implementation of container specification

net.strandbygaard.xflow.security All signature and encryption functionality is implemented by this package.

net.strandbygaard.xflow.utils Assorted functionality, including a number of XML utility classes

net.strandbygaard.xflow.engine Is a client side (mostly) implementation of an engine that processes the container specification

Index

- .Net, 17
 - <container>, 80
 - <document>, 68, 70, 74, 79, 80
 - <ds:Signature>, 70
 - <receipt>, 70–74, 78–80
 - <stepGroup>, 65–68, 70, 72–74, 79
 - <transactionlog>, 62, 69, 71, 79
 - <workflow>, 65, 70, 73, 74

 - Abstract Syntax Notation, 59
 - access, 37, 40
 - access control, 51, 76
 - ACL, 16, 18
 - ActiveCard, 34
 - ActiveX, 75
 - activity, 11, 20–23, 37, 39, 48, 88
 - activity diagram, 26–28, 51
 - AD, 18
 - Adobe, 18, 19
 - Adobe Document Services, 15, 16
 - AES, 40
 - AICPA, 46
 - algorithm, 40, 41, 43, 58, 60, 70
 - asymmetric, 41
 - symmetric, 41
 - Alice, 34, 35
 - Amazon.com, 22
 - AND, 29
 - AND-join, 23, 85
 - AND-split, 23, 29, 85
 - Apache, 77, 78, 92, 100
 - Apache Foundation, 62
 - Apache Lenya, 15
 - Apache Xerces, 76
 - Apache XML Security, 76, 82
 - API, 11, 18, 76, 77, 80, 82, 87
 - application domain, 13, 22, 56, 63
 - application server, 76, 77
 - application state, 51
 - applied cryptography, 40
 - arbitrary, 25
 - ARM, 58
 - ASN.1, 59, 76
 - attack, 76, 83
 - attacker, 76
 - audit, 34
 - audit log, 55
 - audit trail, 62, 89
 - auditing, 16, 55
 - authenticate, 39
 - authentication, 16, 18, 35–37, 39, 41, 53, 54, 100

 - author, 56
 - authorization, 16, 18, 35–37, 53, 54, 62
 - availability, 35, 53, 54
 - available, 36

 - B1456, 44
 - base64, 59, 63, 69
 - basic usage, 82
 - Berkeley, 77
 - binary, 53, 60
 - BitKeeper, 15
 - BLOB, 78
 - Blowfish-128, 58
 - Bob, 34, 35
 - BPEL, 52
 - BPM, 16
 - brute force, 40
 - buffer overflow, 77
 - business process, 20, 22

 - C++, 62
 - CA, 44–46, 83
 - canonicalization, 82
 - CAPI, 76, 80, 81
 - Carol, 34, 35
 - CASE, 28
 - case handling, 22
 - cataloguing, 22
 - cBrain, 15
 - CD-Card, 76
 - cell phone, 57
 - certificate, 53, 76, 82, 83, 97
 - authority, 44–46, 83
 - policy, 44
 - practice statement, 44
 - qualified, 44
 - revocation list, 45, 83
 - certification, 39
 - cipher, 40, 42
 - text, 40, 41
 - Citeseer, 22
 - class, 81
 - class loader, 82
 - client-server, 10
 - CMS, 14, 15
 - compile, 117
 - compromise, 41
 - computational power, 57
 - concurrency, 23
 - concurrent, 23
 - Concurrent Versions System, 15
 - conditional, 22
-

- confidentiality, 35, 40, 41, 55
- container
 - document, 89
- control data, 20, 37, 87, 88
- control flow, 20
- controlled vocabulary, 63
- CP, 44
- CPS, 44
- CRL, 45, 83
- cryptographic hardware, 58
- cryptography, 40
- CSS, 75, 100, 101
- CVS, 15, 60

- Danske Bank, 34
- database, 77, 78
 - schema, 77
- DB, 77
- decrypt, 40, 41
- decryption, 40, 41
- default deny, 36, 76
- denial of service, 83
- DER, 80
- DES, 40
- desktop computer, 57
- digital certificate, 42
- DN, 53, 63, 79, 83, 84, 86
- Document Object Model, 77
- document support, 56
- Document Type Definition, 60
- DOM, 59, 77–80, 82, 87
- Domino Workflow, 15, 16
- DoS, 83
- Doxygen, 92
- DRM, 54
- DSA, 43
- DTD, 60
- Dublin Core, 63, 64

- e-mail, 56
- ECDSA, 43
- Eclipse, 92, 117
- EJB, 76
- electronic document, 38
- electronic documents, 6, 7
- element
 - metaDataType, 63
 - roleType, 63
- EMC, 15, 16, 19
- EMC Documentum, 15, 16
- encoded, 69
- encryption, 40, 41, 43
 - asymmetric, 41
 - symmetric, 40, 41, 58
- endpoint, 62
- EPJ, 33
- ERP, 6
- eToken, 80
- European Commission, 57

- EuroTrust, 44
- exception handling, 22
- exploiting, 57
- external reference, 61

- fabrication, 36
- FESDH, 15
- FileNet, 16
- financial statement, 55
- financial system, 56
- financial transactions, 6
- FIPS 140-3, 34
- Firefox, 75, 101
- firewall, 76
 - policy, 76
- flow control, 22
- FLOWer, 16
- forgery, 39
- FQDN, 64
- FreeBSD, 34
- FTP, 83
- functional test, 114

- GlobalSign, 45
- graphical, 51
 - modelling language, 26
 - notation, 26
- GUI, 82, 96

- hardware, 76
- hash, 42
- hierarchy, 62
- HTML, 101
- HTTP, 64, 76, 82, 83, 101, 102
- HTTPS, 76, 83, 100, 102

- identification, 39
- identify, 39
- identity, 42, 45
- IdM, 32
- IDS, 83
- IETF, 45
- immutable, 89
- impersonation, 36
- implementation, 52
- information security, 35
- infrastructure, 18
- integrity, 35, 42
- interception, 36
- interface, 11
- Internet, 76
- Internet Explorer, 75, 101
- interruption, 36
- intrusion detection
 - system, 83
- invalidate, 62
- invoice, 6, 56
- irrefutable, 42
- IT-infrastructure, 53, 55

- IT-security, 34, 35
- iterator, 117
- Java, II, 17, 59, 62, 75–77, 80, 81, 83, 86, 87, 96, 117
- Java Native Interface, 81
- Javacard, 58
- Javadoc, 78, 92
- Javascript, 101
- JNI, 81
- JSP, 76
- Junit, 113
- JVM, 76
- key, 40–42, 45
 - distribution, 41, 42, 45
 - management, 41
 - private, 75, 76
 - secrecy, 45
 - secret, 40
 - signing, 75
- key pair, 42
- Keystore, 80
- Konqueror, 75
- L417, 44
- LDAP, 16, 18
- least privileges, 45
- legal force, 44
- legal framework, 43
- legislation
 - current, 44
- library, 76
- Linux, 16, 17
- log4j, 83, 84
- logging, 16
- login, 53
- Lotus Notes, 15
- Mac, 16
- malicious code, 32, 34, 57
- Mallory, 34, 35
- man-in-the-middle, 36
- Massachusetts, 57
- MCV, 96
- memory, 77
- message, 40, 41
 - integrity, 41
- meta data, 20
- Microsoft, 18, 19, 60, 76, 77, 80
 - BizTalk, 14
 - Office, 17
 - Outlook, 15
 - Sharepoint, 15
- model checking, 28
- Model View Controller, 82
- Mozilla, 75, 101
- multi choice, II, 29, 65, 80, 85, 88, 89
- multi merge, II, 80, 85, 88
- MVC, 82
- MySQL, 77
- namespace, 71, 77
- National IT and Telecom Agency, 44
- National Security Agency, 40
- NIST, 40, 43
- NSA, 40
- object, 77
- object constraint language, 28
- Object Management Group, 27
- OCES, 6, 44–46, 56, 75, 76, 80
- OCL, 28
- Office, 77
- official document, 39
- OMG, 27
- OpenBSD, 34, 53
- OpenOCES, 81
- OpenOffice, 77
- OpenSign, 81
- OpenSSL, 80
- operating system, 75
- OR, 29
- OR-join, 24, 85
- OS, 76, 77
- OS/400, 17
- OTP, 34
- Pallas Athena, 16
- Palm Pilot, 58
- parse, 77
- parser, 59–61
- passport, 39
- password, 18
- pattern, 81
 - sequence, 18
- PDA, 57, 58
- PDF, 16, 17, 58, 92
- PEM, 80
- Petri Net, 26–28, 51
- PGP, 45
- PHP, 100, 101
- phpCollab, 15
- physical access, 54
- physical domain, 39
- physical machine, 54
- PKCS11, 58, 76, 81
- PKCS12, 76, 80, 97
- PKI, 44–46
- PKIX, 45
- platform, 16, 56
- platform support, 57
- plugin architecture, 81
- policy, 35, 36
- precondition, 31
- price, 16
- private key, 42
- process, 51

- production data, 21, 28, 37, 53, 87, 88
- programming
 - paradigm, 83
- programming language, 76
- protocol, 62, 76, 83
- PS, 92
- public-key
 - authentication, 41
 - encryption, 43
- public-key encryption, 41
- public-key infrastructure, 45
- purchase order, 6
- PwC, 60

- query, 60, 71

- RAR, 56
- RBAC, 16, 18
- RDP, 58
- regular expression, 71
- Relax NG, 60
- replay, 36
- reproduction, 39
- resource
 - allocation, 22
 - constrained device, 57
 - constraint, 57
- resource starvation, 83
- risk analysis, 34
- role, 21, 39–41, 53, 54, 77, 89
- routing, 20, 83
- RSA, 34, 41
- run-time, 52

- S/MIME, 14, 41
- Safari, 101
- SAP, 15
- SAP R/3 Workflow, 16
- SAX, 72, 77
- scalability, 29
- scale, 76
- Scan Jour, 15
- schema, 60–62, 64, 71
- Schematron, 60
- searchable, 77
- SEC, 6
- secure, 7
- secure hardware, 33
- SecureID, 34
- Securities and Exchange Commission, 6
- security, 18, 39, 76, 83
 - model, 41
- security analysis, 31
- security component, 76
- security level, 33
- security model, 18
- security objective, 31, 36
- self documenting, 52
- SEQ, 16

- sequence, 65, 68, 72, 73, 85
- server infrastructure, 57
- Service-Oriented architecture, 83
- servlet, 76
- session binding, 83
 - late, 83
- SGML, 60
- shared key, 40
- signature, 18, 19, 39, 42, 43, 53, 54, 73, 82, 89, 92
 - detached, 61
 - electronic, 44
 - enveloped, 61, 73
 - enveloping, 61
 - scheme, 43, 53, 54
- signature scheme, 53
- Signatursekretariatet, 44
- Simple Object Access Protocol, 83
- Sitecore, 15
- smart-card, 58
- SMTP, 83
- SOA, 83
- SOAP, 62, 83
- Solaris, 16
- source code, 92, 117
- SQL, 77, 100–102
- SSH, 41, 43
- SSL, 83, 84, 86, 87
- Staffware, 16
- standalone, 76
- state diagram, 27
- Subversion, 15
- SUN, 92
- Sun Solaris, 34
- SVN, 15, 60
- swim-lane, 28
- Swing, 96
- synchronization, 23
 - barrier, 23
- synchronizing merge, 29, 65, 89, 112
- system architecture, 56
- system test, 112

- TAR, 56
- TDC, 44, 76
- TDC Certificeringscenter, 44
- terminal
 - dump, 58
 - protocol, 58
- Texas Instruments, 58
- Thinlet, 96
- threat
 - agent, 32, 34
 - level, 34
 - macro, 34, 35, 38
- threat macro, 34
- Tibco, 16
- token, 81
- transaction log, 55

- transaction management, 22
- transition, 20–24
- transition pattern, 18, 21–24, 26–29, 52, 65, 68
 - arbitrary cycles, 25
 - cancel activity, 25
 - cancel case, 25
 - complex, 22, 24, 25, 29, 49
 - deferred choice, 25
 - discriminator, 25
 - exclusive choice, 22, 24
 - implicit transition, 25
 - interleaved parallel routing, 25
 - milestone, 25
 - multi-choice, 25
 - multi-merge, 25
 - multiple instances, 25
 - parallel split, 22, 23
 - sequence, 22, 23, 27
 - simple, 22, 24–28, 49
 - simple merge, 22, 24
 - synchronization, 22, 23
 - synchronizing merge, 25
- trojan horse, 34
- trust, 45, 89
- trust model, 39, 53
 - binary, 53
- type safe, 117
- type system, 71

- UML, 26–28, 51, 112, 113
- unalterable, 42
- unforgeable, 42
- Unified Modelling Language, 26, 27
- unique, 42
- unit test, 113, 114
- Universal Resource Identifier, 82
- Unix, 16, 17, 117
- URI, 61, 78, 82
- URL, 64
- use case, 8, 80, 95
- user account, 75
- user administration, 32
- user directory, 11
- user management, 11
- username, 18

- validate, 60
- Valoris, 57
- verification, 42, 61, 62, 70
- VeriSign, 45
- version control system, 14
- virtual machine, 76
- virus, 34
- Visual Source Safe, 15
- Visual WorkFlo, 16
- VM, 77
- VmWare, 34
- VNC, 58

- VSS, 15
- W3C, 60–62, 71, 100, 101
- web service
 - security, 83
- webserver, 76, 100, 102
- website, 83
- WebTrust, 46
- Windows, 16–19, 34, 58
- Word, 56
- workflow, 10–15, 18–46, 48–52, 54–57, 59–62, 64–66, 70–73, 85, 86, 88, 89, 93, 111, 112
 - engine, 14
 - expressability, 16
 - process definition, 20, 21
 - system, 7, 13–15, 20
- WS-routing, 83
- WS-security, 83

- X-flow, 8, 10, 15, 19, 28, 29, 31, 32, 34, 60, 61, 71, 76, 77, 81, 83, 86, 112, 113, 117
- X.509, 14, 16, 18, 19, 43, 45, 46, 54, 56, 63, 80, 81, 88, 100, 102
- Xen, 34
- Xerces, 59, 78
- XHTML, 75, 100, 101
- XML, 11, 7, 8, 59–63, 68, 71, 72, 76–79, 82, 86–89, 92, 96, 117
 - DTD, 60
 - schema, 60–62, 64, 71
 - schema language, 60
 - Signature, 61, 62, 76, 77, 81, 82, 89
- XML Schema, 76
- XML Schema Language, 60
- XML Signature, 61, 62, 76, 77, 81, 82, 89
- XML-RPC, 83, 86, 88
- xml:id, 62, 67–69, 71–74, 79, 80
- XOR-join, 68
- XOR-split, 24, 68, 85, 112
- XPath, 60, 68, 71, 80
- XUL, 75

- ZIP, 56
- Zope, 15
- zOS, 17