Abstract Document Systems

– instantiated for patient medical records

Master Thesis by

Allan Lindqvist
Brian Christensen

The Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
The Technical University of Denmark

August 2003 – April 2004

IMM-THESIS-2004-23

Abstract

Developing a piece of software for a customer introduces a series of challenges for both developer and customer. One of the key things to be aware of is the language barrier that exist between the two groups – they do not speak the same language. This poses a serious problem to the correctness of the final system as it is likely that misunderstandings lead to a system different from what was intended and needed by the customer. Consequently, this problem exists when developing electronic document management systems. This domain will command extreme attention in the near future as most administration tends to go from paper to electronic documents – powerful and easy-to-use software will be in demand. This Master Thesis addresses the development of an electronic document system platform providing versioning, structuring, document abstraction, and distribution. The development will follow a methodology with emphasis on the domain analysis ultimately leading to a platform supporting an intuitive scripting language originating from the paper document domain easy understandable by both customer and developer, while hiding technical but necessary aspects. The usability of the platform is tested by instantiating it in a domain with a long tradition of paper management – a hospital.

Keywords: *Electronic document management systems, distributed systems, security, domain analysis, document domain, methodological software development, databases, XML, hospital domain, electronic medical records, graphical user interface design*

Resumé

Når der udvikles software til en kunde introduceres en række udfordringer for både udvikler og kunde. En af væsentligste ting, man skal være opmærksom på, er den sproglige barriere, som eksisterer mellem de to grupper – de taler ikke det samme sprog. Det medfører vanskeligheder med hensyn til korrektheden af det endelige system, da det er sandsynligt, at misforståelser kan føre til et system, som er forskelligt fra hvad der var behov for og ønsket af kunden. Heraf følger, at problemet også eksisterer, når der udvikles elektroniske dokumenthåndteringssystemer. Dette domæne vil kræve særlig opmærksomhed i den nærmeste fremtid fordi meget administration har tendens til at bevæge sig fra papir til digitale dokumenter – stærke og let-anvendeligt softwareløsninger vil blive efterspurgt. Denne afhandling omhandler udviklingen af en elektronisk dokumentshåndteringsystem-platform, som tilbyder versionering, strukturering, dokument abstraktion og distribution. Udviklingen vil følge en metodologi med vægt på domæne analyse, som i sidste ende fører til en platform, der understøtter et intuitivt scripting sprog med oprindelse i papirdokument-domænet, som let kan forstås af både kunde og udvikler, alt imens de nødvendige tekniske detaljer gemmes væk. Anvendeligheden af platformen testes ved at instantiere den i et domæne med en lang tradition for papirhåndtering – et hospital.

Nøgleord: *Elektroniske dokumenthåndteringssystemer, distribuerede systemer, sikkerhed, domæne analyse, dokumentdomæne, metodologisk software udvikling, databaser, XML, hospitaldomæne, elektroniske patientjournaler, grafisk brugergrænseflade-design*

One should not increase, beyond what is necessary, the number of entities required to explain anything

Occam's Razor

Contents

List of Figures

Preface

The enclosed report concludes the M.Sc. Thesis by Allan Lindqvist and Brian Christensen carried out at the Institute of Informatics and Mathematical Modelling at The Technical University of Denmark. Professor Dines Bjørner has supervised the project spanning from August 2003 to April 2004.

Objectives

It is the main objective of this report to document the work we have carried out during our Master Thesis project. Two intertwined software development projects have been completed. They will be described along with the motivation and considerations preceding them as well as the conclusions drawn.

A secondary objective involves how to document a software development project from preliminary domain analysis to a running implementation. We intend to present our work – the design and implementation of an electronic document management system instantiated for patient medical records – with a special emphasis on domain analysis and requirements specification. Our ambition is to provide a by-the-book example of software development when following the principles outlined by Professor Bjørner's book 'The SE Book'. Consequently, we have adopted the terminology and structuring of these principles and made use of the RAISE specification language. This means that the reader must possess some basic understanding of these concepts to fully appreciate parts of the technical aspects of the report.

Organization and Prerequisites of the Material

The report is divided into a number of parts each consisting of a series of chapters relating to the same topic. Parts I-III start with introductory chapters while parts II-IV end with conclusions. The introductions and conclusions are non-technical and are intended for any reader who wants to get a quick general idea of what has been attempted and accomplished during the course of the Master Thesis.

I. **Introduction** holds a general presentation of the entire project as well as a study of technologies of interest to the project, such as security issues and possible development platforms. The 'Technologies' chapter assumes that the reader is familiar with the concept of distributed systems and, to some degree, the RAISE specification language.

II. **Document System** presents a full scale software development of an electronic document management system. Traditional development phases are represented, which include domain and requirements engineering, design, and implementation. To fully understand the development process the reader must be familiar with domain descriptions and requirement prescriptions, the RAISE specification language, object-oriented design, and UML.

III. **Medical Record System** describes a software development of an electronic medical record system. The development process is based on the terminology and principles of the electronic document management system – it is an instantiation of it. To understand this process the reader must be familiar with the structuring of domain descriptions and requirement prescriptions, the terminology presented in part II, and to a lesser degree the RAISE specification language.

IV. **Summary** presents a summary of the entire scope of the Master Thesis as well as other spin-off projects, such as business plans and articles regarding the concept.

V. **Appendix** contains all documents deemed unfit for the main report, including complete specifications, selected source code and other detailed information. To understand the majority of the appendices one must be familiar with the RAISE specification language.

Allan Lindqvist,
Kgs. Lyngby, 19th April 2004

Brian Christensen
Kgs. Lyngby, 19th April 2004

Acknowledgments

During the process of creating this Master Thesis there has been contact with several people from different places. The willingness of these people allowed us to base the thesis on real-life information and take the prototype development further. Because of this, the authors would like to thank the following persons and companies.

Healthcare Industry Leader Nordic Hans Erik Henriksen of IBM was the first to provide a vision of the future healthcare IT systems. He introduced the IBM view of healthcare, IBM products for the domain and general insight and opinions on the future to come. Indirectly, this acquaintance combined with the nature of our Master Thesis and the approval by IBM Vice President Kim Østrup got us a ticket to IBMs Student Recognition Event in their development center in Hursley, England.

CEO Jørgen Jørgensen from Rigshospitalet (RH) lead the way for a meeting with selected people from the Danish hospital RH such as the Managing Director of the Heart Center Henrik Eriksen and IT Director Bjarne Kohl. This gave us a contact to a leading user of health IT. As a direct result of this meeting, IT Architect Kasper Weibel Nielsen-Refs, also from RH, was assigned to us. He gave us a good understanding of the problems and successes of RHs 'roll-out' of IT. He also provided an essential contact to Amager Hospital (AH) due to his later involvement with this hospital.

At AH IT Project Consultant Sue Mattoon gave an exclusive session in the use and principles of their note module of an electronic medical record prototype. Furthermore, IT project consultant Jørgen Mikkelsen gave an exclusive presentation of the more technical aspects of the prototype. Finally M.D. Thomas D. Clausen provided invaluable insight into the hospital domain, which included an exclusive interview and access to anonymous genuine paper medical records.

Microsoft's PR Manager Sara Helweg-Larsen provided essential contacts with regards to hardware. She connected us with Hewlett Packard's PR Manager Henrik Kirkeskov who facilitated a loan of a state-of-the-art HP Tablet PC through out the duration of the project. Furthermore, she put us in contact with Product Manager Windows & Windows-Mobile Nis Bank Lorenzen, also from Microsoft, who gave us an introduction to the world of tablet PCs. He approved the loan of yet another HP Tablet PC in his possession.

Associate Professor Robin Sharp from The Technical University of Denmark has been helpful with regards to the security elements employed. Hospital staff members Camilla Christensen, Merete Lelund, Mette Andersen and M.D. Nina Keldsen have given fundamental contributions to the development of the hospital domain. Finally, Rikke Bahr Stidsen and Rikke Hartung have aided with proof-reading.

Part I

INTRODUCTION

Chapter 1

Generally

Large scale computer systems become increasingly more integrated into our everyday lives. It is primarily at work the systems appear but, considering the current pace of development of computer technology, it is not unlikely that software systems will eventually serve in every aspect of our daily routines. As the systems increase in complexity so do the requirements from the customers to the developers – implicitly leading to an increase in software development size and complexity.

The two parties involved in a software project can be simplified to developers and customers, each experts in what they do. The developers are experts in technologies and programming and can combine these two notions to produce functional computer systems. The customers are implicitly experts in their field of work – they are domain specialists.

If there are only skilled people on both sides – developers vs. customers – why do some of the development projects result in systems that do not live up to the expectations of the customers?

The problem lies in the 'interfacing' between the two sides, they cannot communicate properly and even worse they misunderstand each other – they do not speak the same language. This ultimately results in ambiguous and imprecise requirement specifications, which lead to system functionality differing from what was intended and needed. As we see it, there are a number of issues that contribute to this customer/developer gap, the main thing being the parties confronting each other with terms and concepts which are unfamiliar to the other:

1. Creating a computer system calls for certain computer oriented aspects, such as security and data distribution that must be dealt with. Although the customer cannot be expected to understand these issues, decisions on how to address them must necessarily be integrated in the requirements specification and consequently in the dialog between the customer and developer.

2. If the developer does not have sufficient insight in the application domain and the terms associated with it he might not fully understand the requirements and business processes being prescribed by the customer.

3. To structure the requirements and make them ready for design and implementation, a specification language, using a graphical or mathematical notation, is applied which is too abstract and technical for the customer to fully comprehend.

Naturally, it is desirable to somehow reduce the gap between customer and developer and thereby reduce the cost and increase the quality of the produced software systems. In the software development industry several measures are being taken in an attempt to minimize the language barrier:

Customer becomes developer Some software systems can be customized using intuitive graphical user interfaces and simple scripting languages. This makes it possible for the customer to tailor the system to fit his specific needs essentially making him a developer of the system without losing his domain expertise. This principle is a trade-off between making the systems flexible versus enabling them for easy customer modification. As the flexibility of a system increases so does the complexity of customization, eventually making it too difficult for the customer to manage on his own.

Developer becomes customer This is achieved through extensive domain analysis before requirements specification commences. It is the responsibility of the developer to acquire an extensive knowledge of the application domain by spending a considerable amount of time on domain modelling through interviews and ethnography. The domain development will, if executed properly, produce a terminology and a narrative that both the customers and the developers can understand and relate to during the discussion of requirements. For the developers the domain analysis might lead to a formalized model that can be used for further system design and reference when in doubt.

Combination This approach introduces a third party in the software development process. A system framework is delivered which supports the fundamentals of a general domain, such as ERP or content management. The framework contains the building blocks for piecing together a specific instantiation of the domain but it is the responsibility of the third party – solution centers or specially trained customers – to find the right constellation of building blocks which match the customer needs. This concept is used by Microsoft Business Solutions, SAP, and many others. It allows for very flexible standardized frameworks for third party developers to customize into domain specific solutions through dialog with the customer. The third party developers can be considered part framework developer experts, part domain specialists, as they continuously gain knowledge of specialized domains within the general domain, implicitly minimizing the language barrier – they speak the language of both developer and customer.

With these general observations of software development in mind and the ambition to reduce the developer/customer gap as much as possible, we have decided to focus on the domain of document management. Developing electronic document management systems (EDMS) is, in our opinion, an interesting area that will command extreme attention in the near future as most administration tends to move from paper to electronic documents. At the moment there are plans to fully digitize several areas of public administration including all Danish hospitals.

The market already encompasses many manufacturers of turn-key software solutions following the principle of 'Developer becomes customer' as well as several off-the-shelf products geared towards the 'Customer becomes developer' principle. We intend to bridge these principles using the 'Combination' approach by developing a document oriented framework, capable of honoring all basic requirements associated with document management. We will, in particular, focus on the process of moving from a paper oriented working environment to a digitized one.

Minimizing the language barrier between customer and developer will be attempted by letting the EDMS framework 'speak' the language of the customer without limiting the developer in utilizing the benefits of digitalization. We will try to accomplish this by analyzing and modelling the actual world domain of paper document management and projecting this model and the associated terminology to the digital domain. The mechanisms of computer oriented aspects, such as cryptography, will be hidden beneath the terminology and presented as a term or concept recognizable in the paper document domain.

The main assumption is that all fundamentals of paper document management have a digital equivalent and vice versa. In the spirit of Occam's Razor, sticking to these fundamentals should result in a limited number of domain oriented entities, which can be used by the customer and developer as a common basis when discussing what the system is supposed to do.

The usability of the proposed framework will be evaluated by instantiating the framework as an electronic medical record system in the Danish hospital domain. The decision to focus on hospitals is based on the preliminary problem formulation found in appendix A page 119 which has evolved and matured into the more fundamental problem of document management in general. A hospital is a relevant domain to study as it has a long history centered around paper document management and because a nationwide digitalization of the domain is underway. Special emphasis will be placed on how to ensure a smooth transition from paper to the world of computers. We will attempt to minimize frustrations in the transition phase by initially imitating existing business processes of the hospital domain – letting business process re-engineering be a step which can be performed later by reconfiguring the EDMS framework.

1.1 Thesis

The proposed strategy for developing the EDMS framework leads to the main thesis,

> *Adopting the terminology and business processes of the paper document management domain results in an EDMS development platform, which minimizes the language barrier between the domain specialists and the developers,*

which has evolved from the original problem formulation (appendix A page 119) focusing only on the hospital domain. To complete a study of the thesis several aspects must be examined, and they can be summed up in the following conjectural points:

1. It is possible to create a model of the paper document management domain.

2. It is possible to computerize and extend the model where the digital equivalences are identified and used.

3. It is possible to design and create a distributed document system based on the computerized model by combining it with current technologies.

4. It is possible to digitize any specific document domain by tailoring the distributed document system.

5. Using the distributed document system it is feasible and sometimes preferable to initially adopt existing business processes surrounding documents when digitizing a domain.

6. A methodological formal approach results in an effective software development process.

Points 1-3 and some of point 4 will be examined during part II of the report when the EDMS platform is being designed and implemented. Points 4-5 will be examined in detail during part III where we will attempt to instantiate a submodule of an electronic medical record system on top of the EDMS development platform. Point 6 refers to the methods used whenever a domain analysis and requirements specification is developed and is, as such, not an integral part of the main thesis, but rather an interesting observation which can be made in parallel.

1.2 Preparations and Literature

Before commencing the practical work of designing and implementing the EDMS development platform and the electronic medical records system on top of it, some initial research have been carried out to put things into perspective.

In the context of document management we have considered the history of documents and document management up until digitalization began. We have conducted several interviews with software companies, such as IBM and Microsoft, and we have have examined a few of the current EDMS products, including the market leader Documentum [2] and one of the minor but very popular ones, Concurrent Versions System [5]. This study has provided an insight in the functionality currently provided by such systems as well as the terminology they introduce. Furthermore, we have carried out preliminary studies of the current technologies and standards used for storing and exchanging electronic documents, such as ODA (Open Document Architecture) [1, 26, 18, 9, 10] and XML [28, 7]. An overview of this study is presented as an introduction to part II of the report which deals with the EDMS development.

In the context of distributed systems we have examined the principles of distributed programming [4, 30] and the technologies associated with the architecture of such systems. This includes database storage [14], security and communication [33, 32, 3], operating systems and GUI design [27, 21]. The study of these technologies is presented as a technology walk-through in chapter 2 page 9.

In the context of electronic medical records we have conducted several interviews with both doctors, nurses, and IT staff members at hospitals who represent the customers of such systems. We have also had sessions with employees at IBM who represent the developers of such systems. We have studied the current business processes of hospitals [17, 37, 29, 11, 35] and the plans for future business processes after digitalization [36, 13, 34, 25, 12, 22, 19]. An overview of this study is presented as an introduction to part III of the report which deals with the EMR development.

1.3 Delimitation

This being a project of limited time and resources we have been forced to limit the scope of the topics discussed. Some aspects are discarded due to their tedious nature, others because they were deemed less vital or too time consuming. This has resulted in the following restrictions:

- The paper document management domain analysis is based solely on the authors' experiences with documents.
- The presentation of the object-oriented design of the document system is limited to showing overall principles and considerations.
- Document distribution between EDMSs is designed but not yet implemented.
- The domain analysis of medical records, including business processes and document types, focuses on one specific area within the domain.
- The development of the medical record note module prototype is simplified to support only this specific area of the domain.
- The requirements engineering, design and implementation of a G-EPJ compatible client is discussed but not realized.
- The main focus of the prototyping is on proof-of-concept not optimization.

1.4 Contributions

Based on our findings and the work we have carried out we believe that this Master Thesis contributes both technologically and scientifically. Technologically by

- methodologically developing an EDMS framework prototype that incorporates a series of current technologies within the field of databases, XML, and cryptography (part II, chapters 6-9),
- showing an instantiation of the framework by creating a medical record note system, implicitly suggesting how to apply the technologies using tablet PCs and smart cards (part III, chapters 12-14),
- developing a mapping layer that on top of a relational database imitates an XML enabled database (part II, chapter 8),

- developing a common interface for establishing secure network transactions (part II, chapter 6).

Scientifically by

- defining a new concept of documents (part II, chapter 4),
- modelling the domain of paper documents (part II, chapter 4),
- suggesting a new design approach for future electronic document management systems (part II, chapter 9),
- showing an example of the use of a full scale methodological software development (part II, chapters 4-8),
- showing the relationship between specification, design and implementation (appendix I page 192).

1.5 A Note Before Reading

The EDMS framework development documented in this report uses the principles and terminology of 'The SE Book' [6], the RAISE specification language (RSL) [16], and to a lesser extend the principles of [31]. It begins with a presentation of a domain model that initially provides an overview, i.e. a modified truth, of the complete domain. This overview is then gradually extended resulting in the complete model. This means that some elements of the domain analysis might seem incomplete or unmotivated at first, but they will make sense later once the entire model has been established. We believe this approach allows for a more reader friendly introduction to the domain, which, in our opinion, is the most essential part of the Master Thesis.

We knowingly use anthropomorphism – attributing human personality to anything impersonal or irrational, say a computer program – though this, in its strict sense, is incorrect: A program does nothing. But it may prescribe that certain actions are affected by machine – when a machine interprets ("executes") the program text (Glossary anthropomorphism / anthropomorphic from [6]).

Instead of compensating for the fact that developers, customers, etc., may be be of either male or female gender we have decided, for the sake of consistency, to refer to persons as always being of the male gender.

Chapter 2

Technologies

At some point when designing and implementing any distributed system it has to be decided which technologies it is to be based on. All distributed systems share a common base of fundamental requirements in order for them to function:

1. **Hardware Platform** – Hardware platforms for both the client and server application are required.

2. **Operating System and Language** – An operating system running on this hardware is needed as well as a programming language for implementing client and server applications for the operating system.

3. **Storage** – A database is required for storing server information, and a data format is required for encapsulating information when transporting information outside the database.

4. **Security** is required for making the distributed system trusted and safe.

Within these four areas, several technologies can be used to realize the requirements of the distributed system. Which specific technology to choose depends on the given situation. This chapter introduces the four aspects and evaluates different technologies that might be useful once the prototyping of the document system commences.

2.1 Hardware Platform

A typical distributed system consists of one or more server applications residing on stationary hardware platforms combined with one or more client applications running on stationary- or mobile hardware platforms. The communication between servers and client calls for some kind of network, wired or wireless, depending on client types.

2.1.1 Stationary Platform

There are many manufacturers of desktop computers and servers. In general, they all provide the same functionality so the choice of technology boils down to selecting a manufacturer.

2.1.2 Mobile Platform

If mobile clients are required there are several technologies which can be used. They all share the common requirement that wireless networking should be established as wired networking would limit the mobility of the clients severely. The most commonly used protocols for wireless networking are:

- **Bluetooth**, which is a wireless communication protocol for short distances. It was originally developed for electronic devices to be used for data exchange when held next to each other. The standard range for wireless Bluetooth communication is 10 meters.
- **Wifi-compliant** wireless networking also known as the 802.11b standard. Today this is the most commonly used wireless networking technology. It uses a combination of access points to create a wireless network grid of any preferable size, that clients with built in antennas can access. The standard range for Wifi-compliant wireless clients is typically in the range of 35-100 meters.

Mobile hardware is constituted roughly by three different technologies which all offer distinct advantages and disadvantages:

- **Laptops** – This is essentially a complete personal computer made small enough to carry around relatively easy. It has a built-in monitor, which can be of a reasonable size, as well as a keyboard and usually some sort of touch pad or trackball to emulate mouse movement. It is physically the largest of the three, but also the easiest to interact with because of its keyboard.
- **Tablet PCs** – This is also a complete personal computer. It consists of a monitor the size of a pad of paper and possibly a docking station with keyboard. The Tablet PC itself has no keyboard for entering information – instead a pen is used on the touch screen monitor. This pen emulates mouse movement and a picture of a keyboard can be shown on the screen and typed on by clicking the pen on the picture. Handwriting can, to some degree, be recognized automatically as well. The Tablet PC excels in having a large enough touch sensitive screen to show an entire client application while still being more portable than the laptop. It suffers, however, from the lack of keyboard which makes human-machine interaction less efficient, if large amounts of data has to be entered.
- **PDAs** – This is the smallest of the mobile clients. It consists of a small touch screen and pen similar to the Tablet PC, and can only show concentrated information. It cannot be considered a complete personal computer like the other two in terms of speed and hardware interfaces, but it is certainly the most portable as it fits in a jacket's pocket.

2.2 Operating System and Language

An operating system (OS) is required for running the client and server software applications on the hardware platforms. The OS does not necessarily have to be identical for the two – there are some OSs which server software might benefit from while others are more suited for clients. Selecting the appropriate OS is at the present time more a matter of 'religion' rather than functionality. The discussion of which is the 'better' OS has been an ongoing debate for years and the following summary is to be considered the subjective opinion of the authors. Furthermore, the focus is on the most obvious choices of OSs at the moment.

- **Microsoft Windows** – Presently the most widely used OS for personal computers. There are compatible versions which are suited for server and client applications, respectively. Furthermore, compatible special versions of Windows are available on all three types of mobile hardware platforms.

- **Linux** – The dominating 'open source' OS, and as of this writing, it is becoming increasingly popular due the fact that it is free to use both privately and commercially. It has proven stable and well-suited for server applications. It is not as user friendly as Microsoft Windows, which is the main reason why Windows is still the preferred client OS.

- **Web browser** – While not exactly an OS, browsers are increasingly becoming the target platform for client software applications. Either the client applications are embedded within the browser using virtual machine technology, such as Sun Java applets, or the clients are based on the language of web browsers, HTML. The advantage is that most of todays users are familiar with the Internet, and are therefore comfortable working with web pages. Furthermore, it is a relatively future proof platform since it remains basically the same.

Once an OS has been selected a programming language has to be utilized to implement the client and server software applications. Today there is a tendency to use not only a programming language, but an entire development framework as a basis for new applications. Alternatives to the frameworks are basic programming language, such as `C++` or Sun `Java`. These languages provide by themselves enough functionality and libraries to create server applications and graphical user interfaces for clients, but require a lot more standard programming such as network communication and database abstraction layers.

At the moment, two application frameworks, `.NET` and `J2EE`, are becoming increasingly popular when developing distributed systems. The frameworks will not be addressed any further nor compared. It should be emphasized, though, that they provide a basis for developing applications targeted at web browsers for the clients. All other aspects of distributed systems, such as security and database access are hidden within the core of the framework. These frameworks are, in other words, well suited when one wants to focus exclusively on the business logic and client application of the distributed system.

2.3 Storage

The heart of the system is the database in which all information is stored. Scalability is essential as is the ability to perform data mining efficiently. Choosing the correct type of database boils down to the requirements of the internal database model. There are several solutions each suited for different applications.

2.3.1 Data Encapsulation Using XML

No matter which type of database is used for storage in the distributed system some form of data encapsulation is required when transporting information outside the database. Presently, it is considered best practice to encapsulate data using XML technology [7, 28] when transporting it between systems and clients and servers. Extensible Markup Language (XML) is a specialization of Standard Generalized Markup Language (SGML) [8], and has become the technology of choice these days for describing data.

Quoting the W3 consortium "*... the XML syntax uses matching start and end tags, such as <name> and </name>, to mark up information. A piece of information marked by the presence of tags is called an element; elements may be further enriched by attaching name-value pairs called attributes. Its simple syntax is easy to process by machine, and has the attraction of remaining understandable to humans.*"

It is, of course, an alternative to design a unique data encapsulation format to be used exclusively by the distributed system. This, however, could introduce compatibility problems when interfacing to other systems as they would have to accommodate this non-standard type of data representation.

2.3.2 Relational Database

The relational database model (e.g. SQL-databases [14]) is the most widely used database technology. It has shown itself to be scalable and fast – satisfying the needs of the majority of developers in need of database storage. The immediate disadvantage is that the database has to be predefined, i.e. the developer has to know the structures of the data provided to the database in advance. A model of the fundamentals of the relational model is shown in the following:

```
scheme relationalDB =
  class
    type
      FieldName, RowName, TableName,
      Data, Status, database_query,
      Command = database_query

    type
      Row = FieldName -m-> Data,
      Table = RowName -m-> Row,
      DB = TableName -m-> Table

    variable
      db : DB := [ ...tables... ] /* Preinitialized database tables */

    value
      put_data : Data × Command → write db Status,
      get_data : Command → read db Data × Status
  end
```

2.3.3 XML Enabled Database

XML enabled databases are essentially a relational database with an added XML mapping layer used for interfacing to the database. XML documents may be inserted into the database through the mapping layer which separates the XML documents into a structure fitting the tables in the underlying relational database. This mapping from XML document to database fields may result in loss of information depending on the type of mapping, e.g. if the mapping of the data is not complete the surplus of unmapped data is lost. The outcome is that data extraction through the mapping layer, which results in an XML document, might not match the exact XML document originally inserted.

Database manipulation can be carried out using XML queries, such as XPath, and also using a relational database query language, such as SQL, thereby bypassing the XML mapping layer and accessing the relational database directly. A model of the XML enabled database is illustrated in the following:

```
scheme XMLenabledDB =
  class
    type
      FieldName, RowName, TableName,
      Data,XML_doc, Status,
      Command == XML_query_language | database_query

    type
      Row = FieldName →m Data,
      Table = RowName →m Row,
      DB = TableName →m Table

    variable
      db : DB := [ ...tables... ] /* Preinitialized database tables */

    value
      put_data : XML_doc × Command → write db Status,
      put_data : Data × Command → write db Status,

      get_data : Command → read db XML_doc × Status,
      get_data : Command → read db Data × Status
  end
```

2.3.4 XML Native Database

The XML native database is the pure XML database and does not depend on an underlying relational model. Its fundamental unit is an XML document, like the fundamental unit of the relational database is a row in a table. The XML native database preserves the structure of XML documents inserted into it and returns them exactly as they were inserted. In contrast to the relational database, nothing needs to be known about the structure of an XML document before it is inserted. Database modification and extraction is performed using an XML query language, such as XPath. The basic functionality of the XML native database is shown in the following:

```
scheme XMLnativeDB =
  class
    type
      Name,
      XML_doc, Status,
      XML_query_language,
```

```
    Command = XML_query_language

  type
    DB = Name ⤳ XML_doc-set

  variable
    db : DB

  value
    put_data : XML_doc × Command → write db Status,
    get_data : Command → read db XML_doc × Status
end
```

2.4 Security

As described by [3] there are some things to consider in order to make a system secure:

1. Hardware and software on the specific computer should be capable of preserving the secrecy and integrity of data. This includes data exchanges between two tiers in the system.

2. It is essential to be able to determine who is issuing a command, i.e. authentication is vital in order to prevent unauthorized execution of commands.

3. It is important to be able to determine whether the authenticated person has permission in the system to perform the requested command.

These considerations require different security measures, which will be discussed in the following.

2.4.1 Encryption Principles and Protocols

Authentication and confidentiality across networks is realized through different encryption schemes ensuring secure transactions without introducing too much overhead. A protocol for data exchange utilizing these schemes ensures that neither authentication nor confidentiality is compromised.

Asymmetric

The public/private key principle (asymmetric encryption) facilitates confidentiality and authentication. A detailed model of the asymmetric encryption principle can be found in appendix B.1 page 120 – the fundamentals are outlined in the following.

A key pair consists of a private and public key. One of these cannot be derived from the other. In other words, there exists no function which can 'hack' and deduce the counterpart of a given key:

```
axiom
  ∀ (publickey,privatekey):KeyPair •
    ∼(∃ f:HackKey • f(publickey) = privatekey ∨ f(privatekey) = publickey),
```

If two key pairs have identical public keys or identical private keys then the key pairs are the identical:

```
axiom
  ∀ (publickey1,privatekey1):KeyPair,
      (publickey2,privatekey2):KeyPair •
    publickey1 = publickey2 ⇒ privatekey1 = privatekey2 ∧
      privatekey1 = privatekey2 ⇒ publickey1 = publickey2,
```

There exists no functions which can decrypt public key encrypted data without using the corresponding private key:

```
axiom
  ∀ (publickey,privatekey):KeyPair, data:Data •
    ∼(∃ f:HackData • f(Encrypt(data,publickey)) = data),
```

The encrypt function provides both encryption and decryption. All data encrypted with a public key can only be decrypted with the corresponding private key:

```
axiom
  ∀ (publickey,privatekey):KeyPair, data:Data •
    Encrypt(Encrypt(data,publickey),privatekey) = data,
```

It is possible to sign data with a private key – the public key, being inverse, can decrypt the sign and implicitly verify sender.

```
axiom
  ∀ (publickey,privatekey):KeyPair, data:Data •
    VerifySign(Sign(data,privatekey),publickey)
```

Symmetric

The conventional or symmetric encryption algorithm excels by being much faster than asymmetric encryption, as it is computationally much simpler to perform. This explains the often seen combination of the two, utilizing the strengths in each system – initially authentication and encryption is done asymmetrically, and once trust is achieved between the tiers the much faster symmetric encryption can be used instead. A model of the symmetric encryption principle can be found in appendix B.2 page 121 – the fundamentals are outlined in the following.

As suggested by the name, only one key is used in the symmetric encryption scheme. The encrypt and decrypt functions are different from each other, but they rely on the same secret key to encrypt and decrypt data:

```
axiom
  ∀ secretkey:Key, data:Data •
    Decrypt(Encrypt(data,secretkey),secretkey) = data,
```

As with asymmetric encryption there are no functions which do not incorporate the secret key that can decrypt the encrypted data:

```
axiom
  ∀ secretkey:Key, data:Data •
    ∼ (∃ f:HackData • f(Encrypt(data,secretkey)) = data)
```

Protocol for Secure Communication

Based on existing protocols described in [32] and [30] and with the two encryption schemes in mind, a protocol for establishing a secure connection can be derived. The full protocol is available in appendix F.1 page 167, and it illustrates the basic principles which take place underneath many security functionalities used in modern distributed systems.

2.4.2 Digital Certificates

Current security technologies based on the encryption principles mentioned earlier make use of standardized digitally signed key pair containers commonly referred to as 'digital certificates'. These digital certificates are signed and issued by a centralized trusted key distribution center (KDC), such as VeriSign. They can also be revoked by the KDCs if this is deemed necessary. Most modern operating systems will by default trust certain commonly used KDCs and all certificates issued by them.

A digital certificate contains both a public and private key. The public key can be forwarded to other parties inside a certificate container signed by the KDC. This means that you can broadcast your public key to everyone and because the public key will be signed by the generally trusted KDC they can verify that the public key is indeed yours.

Once other parties possess your public key they can use it to encrypt and send messages to you, which you will be the only one capable of decrypting (using your private key in the certificate). In the same way, you are able to sign any message with your private key, and other parties will be able to verify the signature using your public key.

A smart card (credit card sized memory store) could serve as a container for the digital certificate. This utilization could ease the computer interaction regarding system login – using a smart card reader connected to the computer and a entering a memorized password the system could authenticate the user and establish a secure connection using the digital certificate.

2.4.3 Security Roles

The roles required in a distributed system vary greatly and can therefore not be determined before instantiation of an actual system. The essential thing to consider, is the possibility of being able to employ security roles in a fine-grained manner in order to accommodate as many diverse roles as possible. The principle of the digital certificates is able to honor the requirements dictated by [3] with regards to group security within a system. As an alternative, one could consider building the authorization functionality into the core system allowing for perhaps more flexible roles.

Part II

DOCUMENT SYSTEM

Chapter 3

Introduction

This part presents requirements, design, and implementation strategies of an abstract electronic document management system (EDMS). It takes the reader through a detailed software development with all its facets ending up with a prototype of a domain oriented EDMS.

3.1 A Brief History of Document Management

The very first document management technologies were probably the paper clip, the staple, the folder, and the filing cabinet. Before the PC revolution of the early 1980's these were the most commonly used technologies when grouping, categorizing, and archiving paper documents. Consequently, most document oriented business processes were centered around the functionality provided by these.

When the PCs were introduced in the 1980's along came the possibility of managing documents electronically. Initially, in a decentralized way by authoring documents on the individual PCs, storing them on floppy discs, and archiving the discs along with paper documents in filing cabinets. The introduction of LAN shifted the storage of documents towards centralized file servers and in step with the price drop of mass storage the limit of the number of documents that could be stored vanished.

These new technologies were a significant breakthrough for document management – people soon learned to store their information on the server. However, the compilation of information on the centralized servers remained unchecked resulting in an ever increasing bulk of information. This development eventually caused problems, as there was no method for controlling the documents once they were created – after a while it became impossible to tell what was important and what was not. The PC revolution had decreased the document production time, but in the process introduced new problems – the amount of unchecked and unstructured data made reuse of information very difficult due to the lack of efficient methods for finding specific documents.

As networking technology evolved the Internet emerged along with Email technology. This began to change the way people worked with documents. The concept of information sharing was broadened and informal communication using Emails became increasingly popular. While this was another improvement in the ability to have faster information exchange it also resulted in a step further into document chaos. It became obvious that document systems capable of efficiently structuring and retrieving information were needed.

3.2 EDMS Today

The problems related to information overload have spawned several EDMSs. They are often tailored to a specific customer segment, but not to the individual customer – the customer adapts, not the system. The common denominator of the systems is their attempt to manage and structure documents as well as enabling the users to quickly retrieve information hidden within these documents.

Concurrent Versions System [5] can be considered one of the early EDMSs. Though being old and limited it holds the basic notions present in practically every EDMS today – the key feature being versioning of a single document. This enables the user to track the changes (versions) of a given document, always having every version available by the touch of a button. This notion has proven vital in large EDMSs, where a given document might be altered by several independent persons.

The format of the digitally stored documents has been, and probably always will be, a major issue. Structured information in an EDMS has to be of some predefined format, which is of minor significance until it has to be exchanged with a third party 'outside' the EDMS. Document exchange between two parties requires that the receiver is able to access the information and view it as intended. One of the first attempts to overcome this hurdle was ODA [1, 26, 18, 9, 10]. It is an ISO standard that attempts to define every document requirement possible with regards to structure and layout, whereby a common exchangeable format is obtained. It is, however, not very well-supported today. Instead XML [7, 28], descendant from SGML [8], attempts to fulfill the demanding role of uniting information exchange under one banner. Though the principles behind the technology date back several decades the technology is considered fairly new and use of it is not fully standardized with regards to the current EDMSs. The exchange problem is instead partly 'solved' by using application specific document formats such as Word, Excel, or Adobe PDF. This guarantees that information is shown as intended and that it is exchangeable provided that the correct software applications are present in both ends. Metadata containing information about the version, author, etc. usually surrounds this exchangeable format when stored in the EDMS.

The EDMSs today have evolved from simple versioning systems to complex yet flexible contents management systems. The systems incorporate project management and work flow support wrapped in streamlined user-friendly graphical user interfaces. The market leader Documentum [2] goes further in an attempt to close the gap between customer and developer by introducing a user-friendly interface that is supposed to enable the customer to customize the system on his own. Leaving the system tailoring to the customers is a fast growing tendency but it requires extremely intuitive programs and experienced customers. It also limits the degree of possible customization since too many options would complicate the process.

3.3 Our Approach

We intend to develop an EDMS prototype from scratch that provides the basic document management features, such as versioning and structuring. The EDMS will be implemented on top of a generic distributed system architecture providing security, storage, and information distribution.

Business processes and contents of documents will be kept abstract hopefully resulting in a generic EDMS development platform, which can be tailored to a specific document domain in a subsequent development process. In other words, the platform will in itself not be a functioning document management system but a common basis for specific document systems to be built on.

The motivation for our approach is the ambition to minimize the language barrier between developers and customers. This will be attempted by integrating an intuitive terminology into the platform, which is to be used by the customer when expressing specific domain requirements and the developer when implementing the requirements on top of the EDMS platform. As previously indicated, paper document oriented business processes were and still are centered around the physical document management technologies available before the PC revolution. In our opinion, the terminology associated with these technologies (such as grouping documents in folders) is still the most intuitive way of expressing document management.

Based on this assumption we intend to analyze and model the domain of paper oriented document management. The terminology, structure, and business processes of this model will be transferred to the digital domain by finding the digital equivalences of the different elements in the model. An example of such an equivalence is that mailing documents in sealed envelopes corresponds to encrypted information transactions in the digital domain. Some elements may be extended to utilize the power of the digital domain.
Using this approach we expect to find that:

- any paper oriented document operation has a digital equivalent,
- by adopting the unaltered paper document domain a terminology and structure familiar to non-developers will be integrated in the EDMS,
- by keeping document contents and business processes abstract an EDMS platform is designed, which can be tailored to a specific document oriented domain.

Chapter 4

Domain Development

General observations of the actual world of paper document management will be presented initially and will serve as an introduction to the establishment of a domain model – both as a systematic narrative and as a formal specification. This work uses [20], a draft model derived in a former course, which is available in appendix C.1 page 122, as an inspiration. Section 4.12 page 32 holds a glossary which describes specific terms and concepts introduced during the domain development.

4.1 Synopsis

A model of the domain of generalized paper document management describing the entities, structures, and behaviors of actual world paper document management is created. This includes defining terms like places with locations, persons, directories, documents, and dossiers as well as manipulation and structuring of documents. The notion 'domain' will refer to the domain of generalized paper document management.

A complete formal specification of the domain has been created and can be found in appendix D page 128. This chapter provides a textual description of the general observations and ideas which led to the model as well as a transition towards the specification through a systematic narrative. For complete understanding of the entire model and the mechanisms behind it the reader is encouraged to study the full specification in the appendix.

4.2 Stakeholders

Stakeholders are generalized to

Global Administration is the administration of the domain. It keeps track of and uniquely tags all places and individuals in the domain.

Local Administration is the administration of a particular place. It takes care of infrastructure maintenance in the place, such as directory structures, location management and document access privileges.

Person is an individual capable of manipulating documents and dossiers – reading, shredding, archiving, editing, etc.

Third Party is an individual who causes a person to create or manipulate a document or is somehow affected by the fact that documents are manipulated. He does not interact directly with the documents himself.

4.3 Interviews

The authors of this Master Thesis have classified themselves as representatives of all of the domain stakeholders, and the exhaustive phase of stakeholder interviews when determining domain description units has therefore been conducted internally. Consequently, the following model is based on the domain as seen by the authors.

4.4 Intrinsics

The domain consists of places (company buildings or houses – a physical well-defined area within the domain) in which there are:

- persons (employees, residents, thieves, unauthorized personnel etc.)
- physical locations (meeting rooms, desks, drawers, bath rooms etc.)
- a single structured directory (a filing cabinet, an archive etc.). It is possible to have a directory physically distributed within a place, e.g. cabinets on different floors/in different departments, however, these cabinets are considered part of a single large directory of the place.

The domain holds uniquely identified paper documents. Printed on documents is an inseparable combination of layout and information providing an overall meaning. Throughout this domain analysis this will be referred to as document contents.

In addition to documents, the domain also holds uniquely identified dossiers for grouping documents and other dossiers together. Stapling documents together or placing them in a folder is abstracted to placing documents in dossiers. Dossiers within dossiers represents sub-groupings within a larger grouping, e.g. guide cards within a folder.

A document or dossier can be placed in a directory or it can be left in a location at a given place. From there it can be picked up and managed by a person present. If the document is confidential it can be placed inside a locked directory drawer (an index in the directory) locking it with one or several keys. It can then only be unlocked and retrieved by persons who have copies of these particular keys.

Persons are the ones managing documents and dossiers and they can perform a number of commands on those in their possession. These commands include, among others, creating, copying, modifying, shredding, and signing documents. One rule is, that documents and dossiers can only be in the possession of one person, location, or directory in the domain at a time. Therefore, commands (manipulations) cannot be performed on a document or dossier at the same time by two different people. When two people are to work on the same document either a copy is made for one or both of them or they place the document in a location, such as a desk, and take turns in acquiring and reading/manipulating it. A summary formalization of the domain entities and their structures is presented in the following (the domain entity structure is classified as 'system').

```
type
  Document, DocumentID,
  Dossier, DossierID,
  Keys,
  Persons,
  Locations,
  Directory,
  Index,
  DirContents

type
  Directory == mk_dir(DirContents × (Index →m Directory)),

  Place = Directory × Persons × Locations × Keys,
  Places = Place-set,

  M: Command → System → System,

  System′ = Places × DocumentID-set × DossierID-set,
  System = {| w:System′ • wf_system(w) |}

value
  obs_Documents : Person → Document-set,
  obs_Documents : Dossier → Document-set,
  obs_Documents : Location → Document-set,
  obs_Documents : DirContents → Document-set,
  obs_Dossiers : Person → Dossier-set,
  obs_Dossiers : Dossier → Dossier-set,
  obs_Dossiers : Location → Dossier-set,
  obs_Dossiers : DirContents → Dossier-set,
  obs_Keys : DirContents → Keys
  obs_Keys : Person → Keys
```

4.5 Business Processes

Business processes are conducted by the persons and administrators when they are manipulating the entities of the domain. The processes are elaborated below:

Global Administration introduces persons into the domain by supplying newborns with a unique id, e.g. a social security number. This administration also allows for the registration, construction and destruction of new places, i.e. they issue and suspend building permits whereby they introduce to and remove places from the domain.

```
PersonBorn: System × Place × Person × PersonID → System,
PersonDeceased: System × Person → System,
IssuePlacePermit: System × PlaceID → System,
SuspensePlacePermit: System × PlaceID → System,
```

Local Administration is the administration of a single place. They take care of creating and destroying locations, key distribution to persons and directory maintenance (setting up locks, creating and deleting directory indexes).

```
MakeKey: System × Place → System,
DestroyKey: System × Place × Key → System
CopyKey: System × Place × Person × Key → System
RemoveKey: System × Place × Person × Key → System
CreateDirIndex: System × Place × Index* → System
DeleteDirIndex: System × Place × Index* → System
BuildLocation: System × Place × Location → System
DestroyLocation: System × Place × Location → System
```

Person is the document and dossier manipulator. He possesses documents and dossiers which he use in order to accomplish his tasks. He can move between places, but he can only be at one place at a time.

- When a person needs a certain document ...
  - ... with known whereabouts, he moves to the document and acquires it after which he possesses it preventing other persons from accessing and performing manipulation on it.
  - ... with unknown whereabouts he needs to actively search for it at (plausible) locations and inquire other persons for possible whereabouts. Still it is not guaranteed that he will find it and if not it is considered lost to him.
  - ... with directory membership, i.e. it is supposed to be in a certain index in the directory, he moves to the directory to do a search in the index. The document is either present or missing. The latter implies that another person took the document or that it has been wrongfully archived potentially rendering it impossible to find.
  - ... from a dossier in his possession, he opens and searches for the particular document inside the dossier.
- When a person edits a document he ends up with a new document (a version). This document physically replaces the original document and contains the information of the original combined with the changes he just made. The original seizes to exist.
- Archiving documents in a locked directory, requires that the person is in possession of the correct keys (issued by the local administration) and then use them to access the directory. He will then need to keep the keys for later retrieval of the documents.

The variety of manipulations a person is able to perform on the documents and dossiers are listed in the following. These commands are fully described in the complete specification of the domain, which can be found in appendix D page 128.

```
Command = CreateDoc | CreateDos | Copy | Edit | Shred
        | DisposeOfDos | GetDocFromDos | PutDocInDos
        | GetDosFromDos | PutDosInDos | GetDocFromDir
        | PutDocInDir | GetDosFromDir | PutDosInDir
        | GetDocFromLoc | GetDosFromLoc | PutDocInLoc
        | PutDosInLoc | SignDocument | ResetMembership
        | SendDoc | SendDos
```

4.6 Supporting Technologies

The technologies typically available in a document environment constitute

- A scribbling pad, which is the most common and intuitive way of generating documents – it is nearly always available, but the documents produced are often lost, mistakenly thrown away, or difficult to read and understand.
- A combination of computers and printers is the most widely used way of generating documents in companies. Document products are often templates with information about author, date and place.
- A typewriter is less commonly used these days, but still a possible way of producing documents.
- A photocopying machine is a common way of creating a document copy.
- A shredder facilitates a way of destroying documents.
- A waste paper bin is another way of getting rid of, usually, less sensitive documents and dossiers. A waste paper bin can be considered a location which is emptied regularly. Once it is emptied the contents of it is lost.

4.7 Management and Organization

The organization layout of the domain can be of either hierarchical or flat structure. The primary concern in either case is the way a person or administrator is allowed to manipulate domain entities:

Local administration is the only one allowed to perform the local administrative actions described in section 4.5.

Persons are able to access a given collection of documents and dossiers depending on their position within a given company as well as specific company policy.

> **Example:** In a company the boss is the only person with the authority to sign documents, his secretary creates and edits the documents, and the intern is only allowed to copy documents for the secretary. All three of them are persons in the specific place, but each of them have different roles/restrictions when dealing with documents.

4.8 Rules and Regulations

These rules vary with the type of place. Common denominators are listed.

- Paper documents, being physical entities, can be manipulated in any way if a given person can access it. It is therefore often required that access to a given company (place) is restricted (id cards, keys, codes etc.). However, these security restrictions can be circumvented. Circumvention is considered intrusion and may be considered disastrous in some places.

- In the daily work it may be required that certain documents are confidential. This could imply that the document in question is locked away and accessible to only a select few.
- Some documents require proper signatures to obtain validity.
- Obsolete documents are often archived for at least five years before they are shredded.
- Some places may require a certain degree of registration (e.g. quality management) whenever a person has somehow handled a specific document. Registration forms (other documents) should then be filled out with date, person name, etc. when he has manipulated the document.

All these rules and regulations are supposed to be upheld by the persons and local administration of the place.

4.9 Human Behavior

Generalizing human behavior certain common patterns can be deducted.

- There are persons with a good sense of document structuring. This group excels in ordering and maintaining their documents by checking correct placement as well as utilizing dossiers and indexes for easy access and overview of the complete document structure. They rarely lose track of a document.
- There are persons with a poor sense of document structuring. Documents may be lost by the person because they are left at unknown locations. Wrongful destruction and directory index misplacement of documents is also a possibility as well as forgetting to shred sensitive documents.
- There can be unauthorized individuals – thieves, industry spies, terrorists, fired personnel – present as intruders within the place seeking classified information or wanting to destroy hard-to-replace documents.
- A person tends to leave documents for a time interval at a given location, perhaps his desk. During this time the documents are available to all persons who can physically access his desk. They can view and acquire the documents even though they might contain sensitive information or be indispensable. If a document is taken from, say a desk, it is now potentially lost to the original owner who left it there.
- A person can lose one or more of his keys which might require replacement of the lock of a directory. This implies that all persons with keys to this particular lock must acquire new keys.
- It can occur that the documents of two or more dossiers are mistakenly mixed together. It is not guaranteed that the person responsible is able to correctly sort the documents into their corresponding dossiers and even if it is possible it might be a time consuming process.
- Creating documents in handwriting can cause misunderstandings if the contents is misinterpreted or unreadable by other persons.

- Signature forgery can occur.
- Organization roles/restrictions can in some cases be difficult to enforce, since possessing a document implicitly enables the owner to perform all commands on the document even though certain of them may not be allowed for that person.

4.10 A Systematic Narrative

The following list of statements express the previous descriptive outline in a systematic and unambiguous way defining the explicit domain model. Figure 4.1 tries to graphically clarify the entity structure described in the intrinsics section. All references below point to the formalized domain specification in appendix D page 128 and are of the structure <page>.<line> [− <line>][+ <line>]. Words that are *emphasized* can be found in the glossary page 32.

1. The *global adm.* can (133.23 - 26)
   (a) introduce *persons* to the domain by registering them with an id.
   (b) remove *persons* from the domain by removing their registered id.
   (c) introduce *places* to the domain by registering them with an id.
   (d) remove *places* from the domain by removing their registered id.
2. The *local adm.* can (133.28 - 35)
   (a) make a new *key* (and corresponding lock).
   (b) destroy a *key* (and corresponding lock).
   (c) copy a *key* from the key repository of the *local adm.* and hand it over to a *person.*
   (d) remove a *key* from a *person.*
   (e) create a *directory index.*
   (f) delete a *directory index.*
   (g) lock a *directory index.*
   (h) register (and build) a *location* within the *place.*
   (i) unregister (and destroy) a *location* within the *place.*
3. A *document* is an abstract entity which can contain any type of information – contents (128.32 + 39).
4. From a *document* one can observe its: unique identifier, time of creation, *document type*, creator, place of origin, *signatures*, *directory membership*, *ancestor* and contents (128.36 - 44).
5. A *document* is either of type *master*, *copy* or *version* (128.6).
6. A *dossier* is a container holding zero or more *documents* and zero or more *dossiers.*

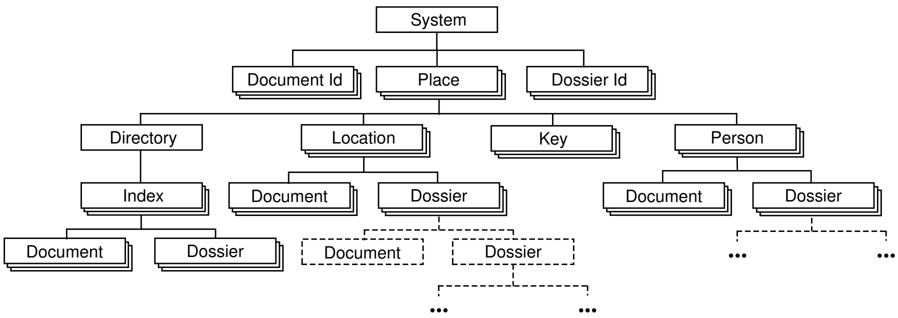


Figure 4.1: Domain Entities Structure

7. From a *dossier* one can observe its unique identifier, a description, and the *documents* and *dossiers* inside (129.46 - 49).
8. A *directory* is a hierarchy of named *indexes* (128.25).
9. A locked *directory index* requires that the *person* wanting to access it possesses the needed *keys* (141.334 + 349 + 367 + 382).
10. A *location* is a container holding zero or more *documents* and zero or more *dossiers*.
11. From a *location* one can observe the *documents* and *dossiers* in it (129.57 - 58).
12. A *place* consists of (133.14)
    - a single *directory*,
    - zero or more *persons*,
    - zero or more *locations*,
    - zero or more *keys*.
13. The domain consists of (133.6)
    (a) a number of *places*,
    (b) a collection of ids which uniquely identify all *documents* in the *domain*,
    (c) a collection of ids which uniquely identify all *dossiers* in the *domain*.
14. A *person* is a container holding – or possessing – zero or more *documents*, zero or more *dossiers*, and zero or more *keys*.
15. From a *person* one can observe his unique id, *keys*, *signature*, *documents* and *dossiers* in his possession (129.51 - 55).
16. A *person* can (all commands involving an existing *document* and/or *dossier* assert that it/they are in his possession)
    (a) create a *document*, after which it is in his possession (138.165).

(b) create a *dossier*, after which it is in his possession (139.189).

(c) copy a *document*, after which the *copy* is in his possession (139.206).

(d) edit a *document* (139.235).

(e) put *documents* or *dossiers* in a *directory index*, provided he has access to the particular *directory index* (141.344 + 377).

(f) get *documents* or *dossiers* from a *directory index* provided he has access to the particular *directory index* (141.328 + 361).

(g) get *documents* from a *location* (142.394).

(h) get *dossiers* from a *location* (142.428).

(i) put *documents* in a *location* (142.413).

(j) put *dossiers* in a *location* (143.447).

(k) delete the *membership* of any *document* in his possession (144.512).

(l) sign any *document* in his possession using his *signature* (143.462).

(m) send any *document* or *dossiers* in his possession to another *person* in a specified *place* (143.474 + 493).

(n) shred any *document* in his possession (140.268).

(o) dispose of any *dossier* in his possession (140.258).

17. A newly created and unedited *document* is of type *master* and has no default *membership* to any *directory* (138.173 + 177 + 178).

18. When a *document* is placed in a *directory index* for the first time, possibly via a *dossier*, it is 'stamped' with the destination *directory index* (*membership*). If a *membership* is already stamped on the *document* nothing happens (134.57).

19. An edited *document* becomes a *version*, if not already, and loses any *signatures* present on the original *document*, but maintains *directory membership* information (140.243 + 246 - 248).

20. A *document* being edited seizes to exist and is replaced by the new *version* of the *document* (140.253).

21. When copying a *document* the new *document* is of type *copy* and does not have a *membership*. From a *copy* one can observe the *document* id of its *ancestor*, which is the original *document* from which it seems to be copied. (139.215 + 219).

22. The *ancestor* of a *document* can only be a *master* or a *version* of an original *master*. If a *copy* is made of a *copy* then it will inherit the *ancestor* of the *document* from which it was copied. This means that a *copy* of a *copy* of a *master* has same *ancestor* (139.221 - 225).

23. *Documents* and *dossiers* are sent in sealed *envelopes* between *persons*. This correspondence is considered secure, i.e. no one else can spy on the information while it is inside the envelope and being delivered (143.474 + 493).

4.11 Formalization

A complete formal specification of the domain model is available in appendix D page 128. The specification is divided into a series of RSL-schemes, an inheritance structure, each isolating a certain aspect of the complete specification:

- `DocSysTypes.rsl` (appendix D.1 page 128) – defines the basic types as well as generic observer functions of the domain model.
- `DocSysBasics.rsl` (appendix D.2 page 129) – introduces operator overloading of different types in order to facilitate an easy manipulation of the model entities. The file also contains helper functions.
- `pDocSysTypes.rsl` (appendix D.3 page 133) – represents the specific domain types and specific domain observer functions.
- `pDocSysBasics.rsl` (appendix D.4 page 133) – specifies domain helper functions.
- `pDocSysWF.rsl` (appendix D.5 page 135) – contains the well formed criteria of the model. These include that a document / person can only be in one place at a time. Consult the specification for further understanding of these criteria.
- `pDocSysCmds.rsl` (appendix D.6 page 136) – describes the commands available to the persons.

The system definition (the domain entities structure), complying with assumptions 8, 12 and 13 and ensuring well-formedness, dictates that,

```
System′ = Places × DocumentID-set × DossierID-set
System = {| w:System′ • wf_system(w) |}
Place = Directory × Persons × Locations × Keys
Directory == mk_dir(DirContents × (Index -m-> Directory))
Persons = PersonID -m-> Person
Locations = LocationID -m-> Location
Keys = Key-set
```

The 'Edit' command, that complies with assumption 16(d), 19 and 20, will be presented in detail in the following:

```
M: Command → System → System
M(cmd)(places, docids, dosids) ≡
  case cmd of
    mk_Edit(person, plid, time, document, (te,fe)) →
      let (dir,pers,locs,keys) = places(plid) in
       assert(person ∈ rng pers ∧
           document ∈ obs_Documents(person));
```

It asserts that the person doing the editing is in fact physically present in that place and that he is holding the document in question.

```
        let doc:Document •
          obs_ID(doc) = obs_ID(document) ∧
          obs_Time(doc) = time ∧
          obs_Contents(doc) = te(obs_Contents(document)) ∧
          obs_Type(doc) = version ∧
          obs_Creator(doc) = obs_ID(person) ∧
          obs_PlaceID(doc) = plid ∧
```

```
          obs_Signatures(doc) = {} ∧
          obs_DirMembership(doc) = obs_DirMembership(document) ∧
          obs_PlaceMembership(doc) = obs_PlaceMembership(document) ∧
          obs_Ancestor(doc) = obs_Ancestor(document)
```

When editing a document a new document is created with almost the same attributes – contents is new and signatures are no longer valid and are therefore cleared.

```
        in
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              (person \ {document}) ∪ {doc}],
           locs, keys)], docids, dosids)
        end
      end
  end
```

Finally the system is updated by removing the original document and inserting the revised document from and to the person.

4.12 Glossary

Ancestor This describes the identification of the *document* from which a given *document* seems to be copied.

Command can be one of the following: CreateDoc, CreateDos, Copy, Edit, Shred, DisposeOfDos, GetDocFromDos, PutDocInDos, GetDosFromDos, PutDosInDos, GetDocFromDir, PutDocInDir, GetDosFromDir, PutDosInDir, ExportDoc, SignDocument, ResetMembership, ReturnDoc, ReturnDos, SendDos or SendDos. In other words, it is the possible manipulation that can be done on a *document* or *dossier* via the model.

Copy *Document Type* choice – describes that the *document* is a copy that has not yet been edited.

Directory A *directory* is a hierarchy of named *indexes*. The hierarchy structure is maintained by the *local adm.*, i.e. they create and delete the *indexes*.

Document Represents a generic document with undefined contents. The attributes unique identifier, time of creation, *document type*, creator, *place* of origin, *signatures*, *directory membership*, *ancestor* and contents are always available for observation.

Document Type The document type is always equivalent with one of the following three notions: *master*, *copy* or *version*.

Dossier A dossier is a container of other dossiers and *documents*, i.e. it is a way of structuring *documents*. It represents a folder, paper clip or other means of grouping *documents*.

Envelope The envelope is a reasonably secure way of sending confidential *documents*, i.e. a transport container that prevents *persons* from spying on the contents.

Global Adm. Stakeholder who introduces and removes *persons* and *places* to and from the model.

Index The smallest unit in a *directory* which contains zero or more *documents*, *dossiers* and other *indexes*. *Local adm.* can equip an *index* with zero or more locks preventing access for *persons* without proper *keys*.

Local Adm. Stakeholder who introduces and removes *locations* to and from a *place*. He also manages *key* distribution and *directory* structure.

Location A location represents a well-defined physical area (excluding *directory* and *persons*) that can contain zero or more *documents* and *dossiers*.

Key Keys provide access to a locked *directory index* provided that it is the correct key(s), i.e. different keys open different locks. Keys are distributed and revoked by the *local adm.*. The *local adm.* has at all times every key to every lock at the *place*, i.e copies are made of the master *key* which is always available to the *local adm.*

Master *Document Type* choice – indicates that the *document* is an original *document* that has not yet been edited – all newly created *documents* are masters.

Membership The membership of a *document* indicates if there is a particular *directory index* at a particular *place* in which the *document* belongs.

Person A stakeholder who also is a container of *documents* and *dossiers*. He is the manipulator of *documents* and *dossiers* in the domain.

Place A well-defined physical area that contains one *directory*, one or more *locations*, zero or more *persons* and zero or more *keys*.

Signature Represents the legally binding signature associated to a given *person*.

Version *Document Type* choice – indicates that the *document* has been edited into the current state.

Chapter 5

Requirements Development

Based on the previously derived domain model the requirements of a domain oriented electronic document management system (EDMS) will be described. As mentioned in the introduction it is intended to digitize the domain with an absolute minimum of business process re-engineering. In fact, it will be attempted to project the entire domain model to the digital domain and only make modifications when it is possible to utilize the digitalization to prevent unwanted human behavior and non-determinism. The terminology of the domain (such as documents, persons, directory, etc.) will be re-used when prescribing the requirements. However, these entities will now refer to digital entities corresponding to the physical objects of the domain.

A complete formal specification of the requirements of the EDMS has been created and can be found in appendix D.1 page 128. This chapter is a structured textual supplement to the specification, which primarily describes the transition from domain to requirements. Section 5.7 page 46 holds a glossary, which describes specific terms and concepts of the requirements being prescribed. For further details about the mechanisms behind the EDMS, the reader is encouraged to consult the full specification in the appendix.

5.1 Stakeholders

The stakeholders of the EDMS constitute

Administrators maintain settings for users (creation, deletion, keys, etc.) and the directory structures (the local administration in the domain).

Users are the ones manipulating documents and dossiers using the EDMS. All users shall be associated with a unique person in the system reflecting their state (possessed documents, dossiers and keys) – this person is controlled by the user, who through him decides how the system is manipulated. The user shall be able to associate himself with this 'virtual' person securely through client software and using a login and password (the persons in the domain).

Maintenance maintains the entire hardware infrastructure, which includes servers, network grid, client computers etc.

Third Party is an individual who causes users to create or manipulate documents or is somehow affected by the fact that documents are manipulated. He does not interact directly with the documents himself.

Foreign system is a computer system of unknown configuration. It either wants to retrieve information from the EDMS or the latter seeks information in the foreign system.

5.2 Business Process Re-Engineering

Entering the digital domain poses some interesting possibilities which were not available in the paper document domain – all information can now be centralized in a server.

- The centralization of information minimizes the need of having to remember things such as document placement and directory keys. However, it also results in dependency on the server, i.e. if the server is down it cuts off access to all documents.
- Users do not need to be present within company buildings to access documents in the system. This means that users can access the company directory at home or from abroad as long as a network connection to the system is available.
- It is feasible to perform encryption when sending data in order to prevent outsiders from accessing classified information. It is now up to the system to establish secure transmissions between persons and not the responsibility of the sender (in the domain the sender would normally use a sealed envelope and a courier).
- If a document or dossier has a membership then it can be returned to it at any time, even if the membership points to a dossier currently in the possession of another person. This is possible because dossiers and directories are now digital concepts as opposed to physical entities in the domain.
- When a document or dossier is removed from where it has membership a reference (ghost) can automatically be left behind indicating that it has been removed. This ghost can also contain information about the person who is currently in possession of the document or dossier.
- Documents and dossiers can be read and browsed, even if they are not in your possession, as long as you own the keys required for retrieving them from the container (index or dossier) where they have membership. Modifying, however, still requires ownership of the document or dossier as this is generally a good way to prevent conflicts.
- Documents are no longer physically replaced during editing. This means that different versions of the same document can co-exist making it possible to extract a version history from a document. A collection of versions, including the master or copy document from which they were created, will be referred to as a document group and documents within the same group

will always be kept together. Examples of document groups are shown later in figure 5.2 page 43.

- As opposed to the paper domain it is now possible to describe how document contents $\mathcal{C}$ is generated on the basis of dynamic data $\mathcal{D}$ (the information entered by the user) with the aid of a template transfer function $f(x)$ that dictates layout as well as defining the meaning of the dynamic information by associating it with static labels. The relationship can be expressed by

$$f(\mathcal{D}) \rightarrow \mathcal{C} \qquad \wedge \qquad f^{-1}(\mathcal{C}) \rightarrow \mathcal{D}$$

  To easily define any kind of document contents $\mathcal{C}$ the following formalized structure can be used. It describes how a template can be defined and how it relates to contents $\mathcal{C}$ and data $\mathcal{D}$:

```
1  scheme contents =
2    class
3      type
4        layout, text, binary, ref,
5
6        txt_label = layout × text,
7        bin_label = layout × binary,
8        txt_input = layout × ref,
9        bin_input = layout × ref,
10
11       𝒞 = group × 𝒟,
12       𝒟 = (ref →m text) × (ref →m binary),
13
14       group == mk_grp(label* × input* × group*),
15       label == mk_ltxt(txt_label) | mk_lbin(bin_label),
16       input == mk_itxt(txt_input) | mk_ibin(bin_input)
17
18     variable template : group
19
20     value
21       f : 𝒟 → read template 𝒞,
22       f^-1 : 𝒞 → 𝒟
23
24     axiom ∀ d : 𝒟 • f^-1(f(d)) = d
25   end
```

  The transfer function contains detailed information about the entire document layout and how and where dynamic data shall be placed on top of the template. The template associates layout to the dynamic data through references (line 12). This interpretation of contents provides the possibility of exchanging templates on the fly and in the process redesign the way dynamic data appears and in which context. The main focus of the template derivation is to separate the entities $\mathcal{D}$ and $f(x)$ in order to be able to store the information in a computer as well as displaying it correctly again when retrieved. The transfer function can be integrated in a GUI or as a transformation of contents one step before presentation in a GUI. Creating contents specifications are kept out of the requirements at this level and left for further specification when the system is instantiated for a specific domain.

The domain-to-business re-engineering operations that are required are presented in the following.

5.2.1 Supporting Technologies

It is now required that every document created, intended for the system, is of digital format. This eliminates the use of scribbling pads and typewriters:

- Computers (desktop, laptop, pocket, palm, cellular etc.) should be the only way of creating documents intended to be part of the system. It is also required that the computer is equipped with specialized software, which must be used in order to guarantee the consistency of the EDMS.
- Printers should still be available if hard copies (exports) are required – event logging and ID tagging of the export is to be carried out.
- Photocopying machines should still facilitate a possible way of duplicating exports, but they should implement a strict level of security, i.e. log the copy event and possibly prevent the copy command if the correct permissions are not possessed by the person doing the copying.
- The shredder is still a way to eradicate physical documents (exports). The machine should possibly be able to log the shredded document ID and log it as erased from the physical realm.
- A centralized server is required. Tight security has to surround it, this being the place where all documents now reside.

Once documents are exported out of the system (to paper) the system cannot be held responsible for securing them – this is left to the exporter, who is held responsible for the whereabouts of his exports until they are registered as shredded.

5.2.2 Management and Organization

Upholding the access privileges is now left to the system. It is possible to introduce single command permissions (using keys) preventing a person from performing certain commands he is not cleared for, such as printing and directory access. Ultimately the access permissions are left to the system and not the person thereby improving enforcement of management and organization guidelines.

5.2.3 Rules and Regulations

Desirable common denominators of this topic can be re-engineered as follows:

- Places are no longer physical, but virtual, so you can potentially access a place from anywhere in the world via your computer. This calls for a strict network protocol for data exchange for guaranteeing authentication of a given person and maintaining security integrity.
- Archives of documents do not require the physical space of an entire basement anymore, but can be kept on a single server or on one or more CD-ROMs, DAT tapes or other backup media.
- Backups can be made of all documents and kept on backup media outside the company. In case of fire, theft, etc., the documents can be restored from such a backup thereby minimizing loss of information.

- Signatures can be made digital – a bit different in handling – but still just as legally binding.

5.2.4 Human Behavior

The migrated system can prevent several undesirable human behaviors of the domain

- With the physical realm gone the user has to change the normal way of managing documents. It is no longer possible to physically spread out documents on a desk or take them with you in the hallway. If such behavior is required the relevant documents must be exported to the paper based domain using a printer.

- It is no longer possible to leave documents in the open. A document is always on a person or in a directory implicitly protecting documents better and preventing unauthorized access.

- The user can not lose a document, meaning that he can always locate a document from its id, or do a contents dependent search for it.

- The system can help with directory and dossier membership, preventing that documents originating from different dossiers are mixed by accident.

- The user can not lose a directory index key.

- Handwriting is potentially eliminated in the system effectively removing the risk of misinterpretation of document contents.

- The user cannot physically shred a document or permanently dispose of a dossier. Instead deletion can be simulated electronically by transferring the document or dossier to a recycle bin where they can be restored by an administrator. This can prevent accidental deletion of information.

- The system can prevent wrongful archiving, if the document in question already has a membership to a different dossier or directory index.

- Persons have to use a special software and comply with the restrictions this encompasses. This means education of users who are inexperienced with IT.

- The digital signatures can be cryptographically associated with the contents of the signed document in such a way that tampering of the contents will void the signature rendering forgery impossible.

- Automatic event logging of a document is feasible facilitating a detailed event history reflecting the whereabouts of and commands performed on a given document. This could serve as a way of discouraging persons from accessing documents they should not be reading even if they do have access to them. On the other hand, it could also be considered as too much surveillance, which is why it should be kept optional.

5.3 Domain Requirements

Performing the domain-to-requirements operation yields well-defined categorized requirements prescriptions, hence the following five sections constitute the complete set of domain requirements.

5.3.1 Projection

The system is limited to the basic functionality of managing documents and maintaining the access privileges to them and the commands. The system maintenance and the notion of the global administration is disregarded. The close relationship to the domain is reflected in the number of domain assumptions (A) adopted without modification – please refer to section 4.10 page 28. All references below point to the formal specification in appendix D page 128 and are of the structure <page>.<line> [− <line>][+ <line>]. Words that are *emphasized* can be found in the glossary (page 46).

1. A.2 (local adm. = *administrator*) – extended later.
2. A.3
3. A.4
4. A.5
5. A.6
6. A.7
7. A.8
8. A.9
9. A.12 – extended later.
10. A.13 – extended later.
11. A.14
12. A.15
13. A.16abcdefklm (assuming the *user* has *permission*) – extended later.
14. A.17
15. A.19
16. A.21

The following assumptions are projected away:

- A.1 (global adm. is not a part of the system)
- A.10 (*locations* are no longer document containers)
- A.11 (. . . and you can therefore not observe anything from them)
- A.16ghij (. . . or perform any commands relating to them)
- A.18 (is re-engineered to having a more elaborate membership function)
- A.20 (is re-engineered to not lose version information)
- A.22 (is re-engineered to reflect the actual ancestor)
- A.23 (is re-engineered to always support secure information exchange)

5.3.2 Determinism

Some non-determinism of the domain can be eliminated:

17. It shall not be possible to send a *document* / *dossier* to a non-existing *person* at a given place (163.580 + 610).
18. It shall not be possible to violate a *membership*, i.e. perform wrongful archiving in *index* or *dossier* placement (ex. 160.386 - 388).

5.3.3 Instantiation

Since the EDMS model is a generalization certain domain specific elements, such as document contents and user interface requirements, shall remain underspecified. Filling in these blanks will be referred to as instantiating the document system for a specific domain. The amount of work required to fill in these blanks and achieve the desired functionality is an indicator of the strength of the EDMS and the model it is based on.

Instantiating the document system shall require the following domain specific extensions to the existing model:

- definitions of the documents and their contents. This involves creating contents templates representing the different kinds of documents of the target domain. Besides being an extremely easy way of adopting existing business processes it helps in determining the dynamic data of documents, which is the only information needed to be stored.
- definition of the stakeholders and their business processes, which dictates the system business logic. This includes describing the supporting technologies, management and organization, rules and regulations, and human behavior, which influence the business processes thereby affecting the business logic.
- a user interface that complies with the contents definitions and business logic.

To ensure a reasonable degree of flexibility it shall be possible to instantiate more than one domain on top of the same document system base. This is desirable when two domains share some of their information but require different user interfaces and/or business logics, e.g. different departments within the same company. Such domains shall be referred to as sub-domains within the same domain.

5.3.4 Extension

There are several extensions possible, feasible and desirable when taking the domain into the computer as illustrated in the former re-engineering section.

19. Each *user* shall have an unambiguous connection to a *person* in the system.
20. To every *person* a password shall be associated that allows only for the *user* with matching id and password to access the specific virtual person in the system.
21. Provided that a *person* owns the required *keys*, he shall also be able to

(a) merge a *document* (165.721).

(b) remove any *document* in his possession to the *recycle bin* (159.336).

(c) remove any empty *dossier* in his possession to the *recycle bin* (159.355)..

(d) export a *document* to a physical *location* (162.528).

(e) set *permissions* on a *document* (164.659).

(f) set *permissions* on a *dossier* (164.682).

(g) return any *document* or *dossier* to its *membership* container (*dossier* or *index*), even if this (*dossier*) container is currently in the possession of another *person* (165.695 and 165.709).

22. Provided that he knows the id and owns the required *keys*, a *user* shall be able to read but not modify any *document* or *dossier* even if it is not in the possession of his virtual *person*.
23. *Administrators* shall be able to

(a) restore *documents* from the *recycle bin* (147.112).

(b) manipulate the system without being bound to regular rules (147.114).

24. The *system* shall be extended with a collection of ids which uniquely identify all *exports* in the *system* (145.6).
25. Each *place* shall be extended with a single *recycle bin* (145.15).

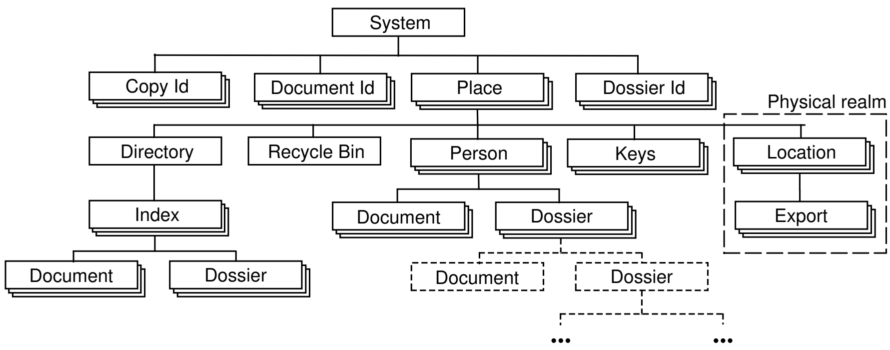


Figure 5.1: Electronic Document Management System

26. It shall be possible to locate a *document* / *dossier* from its id (165.748 + 751).
27. Every *document* shall be a member of a *document group* (145.35).
28. It shall be possible to see a *document history* of a given *document* (165.754).
29. *Membership* is extended to reflect *dossiers* as well (145.24 + 29).
30. If a *document* or *dossier* is removed – not deleted – from where it has *membership* a *ghost* of it shall be left behind.
31. If the *membership* of a *document* or *dossier* is deleted then the *ghost* at the *membership* shall be deleted as well.
32. If a *document* or *dossier* with a *membership* is sent from one *person* to another then the *ghost* at the *membership* container shall reflect this change of owner.

33. A *user* shall not be allowed to perform a *command* which is locked with a *key* his virtual *person* does not possess (152.319 + e.g. 157.262).
34. All *commands* are performed on *document groups*, with the following exceptions
    (a) 'CopyDoc' and 'Export' can be performed on single *documents* in a *document group*.
    (b) 'Sign' and 'Edit' can be performed on only the newest *document* in a *document group*, i.e the latest *version* (158.301 + 162.561).

    This implies, that get/put *commands* shall move the entire *document group*, and that a *document group* shall never be divided.
35. An *event history* shall be available for every *document* – this implies that every *command* performed on a *document* is logged together with relevant information regarding the *command*. This feature shall be an option (145.23).
36. All *document* editions shall be preserved, i.e. a new *version document* is created for every edit (159.327).
37. A *foreign system* shall be able to request from and provide information to the system via predefined queries and data formats.
38. It shall be possible to 'merge' *document A* from group G_A with *document B* from *document group* G_B into *document C* provided that (consult figure 5.2 for a graphical interpretation):
    - *document A* is a *copy* or a *version* of a *copy*.
    - B is the newest *version* in group G_B (165.725).
    - the *ancestor* B_{anc} of the *copy* in group G_A is a *document* in group G_B (165.724).
    - the involved documents obey (can be discretized to being parts of involved documents) (165.726)

$$Assert(B == A \vee B == B_{anc})$$

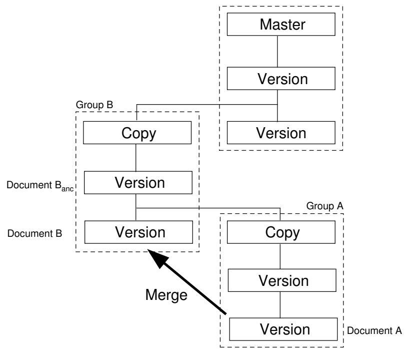
39. When merging *document A* with *document B* all *documents* contained in group G_A are removed (165.724). This is shown in figure 5.2(d).
40. It shall be possible from an *export* to observe a unique export identifier and the id of the *document* which was exported (162.539).

5.3.5 Fitting

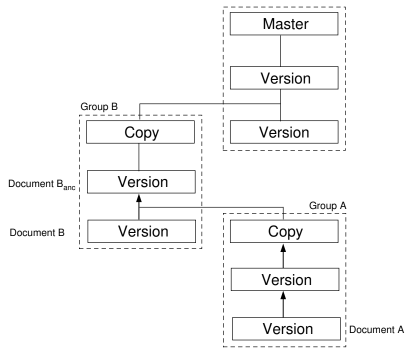
The generalized document system needs fitting but, as stated earlier, it is left to the specific instantiation to fit the system if needed. This requirements document seeks to prescribe a fundamental and general EDMS.

5.4 Interface Requirements

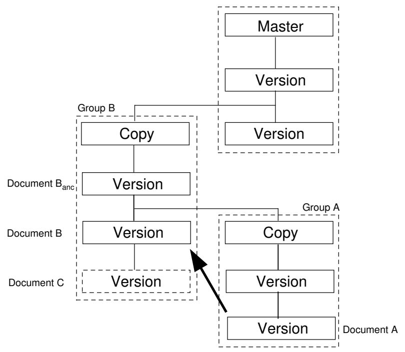
41. A *user* shall be able to log on the system via id and password matching his system profile after which he associates himself with his unique virtual *person* in the system.



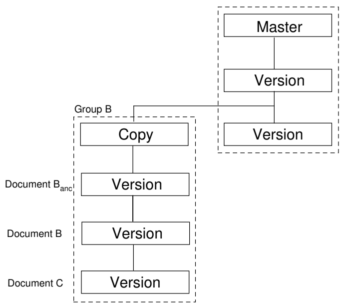
(a) Document A is to be merged with document B. It possible because A is a version of a copy and B is the newest version.



(b) It is verified that the ancestor of the copy in group A is contained in group B.



(c) The documents are merged into C provided that the merge condition is obeyed.



(d) Group A is removed.

Figure 5.2: Merging of Documents in The EDMS

42. A *user* shall via client software be able to
    - perform all valid *commands* through his virtual *person*.
    - view system states, i.e.
        – show information in their relevant context, such as descriptions and contents.
        – see a table of contents (TOC) of relevant data sets, such as *documents* and *dossiers* in a *dossier*.
    - enter information into the system.
    - browse the *directory*.
    - browse *documents* in his possessions.
    - locate a *document* from its id.
    - generate a *document* history from its id.

43. An *administrator* shall via administration software be able to
    - create and edit user profiles, user *permissions* and *directory indexes*.
    - restore *documents* from the *recycle bin*.
    - manipulate the system via direct access.

5.5 Machine Requirements

44. A network shall be available.
45. A proper dimensioned server is required for decent operations – dependent on number of *users*.
46. The server room shall have tight security.
47. The server shall have an extensive automated backup feature.
48. A number of computer devices – PDAs, Tablet PCs, laptops, desktops – shall be available for the client software in order for the *users* to access the system.
49. Printers shall be available if print-outs are required.
50. Shredders shall be equipped with special scanners in order to log shredded exports.
51. Photocopy machines shall be equipped with means of identifying persons and possibly prevent and / or log the event.
52. All data exchanges shall be encrypted.
53. Using cryptography the tiers in the system shall authenticate each other and establish a tunneled communication session.
54. The *foreign system* data exchange shall be based on existing standards (such as XML).
55. The system shall be scalable with regards to clients, data size and distribution of directory.
56. All data shall be stored in one or more databases.
57. Signing documents shall adhere a respected digital signature protocol guaranteeing integrity.

5.6 Formalization

A complete formal specification of the domain model is available in appendix D. The specification is divided into a series of rsl-schemes, an inheritance structure, each isolating a certain aspect of the complete specification:

- `DocSysTypes.rsl` (appendix D.1 page 128) – defines the basic types as well as generic observer functions on types for a document system. This specification is also the basis of the domain, hence the basic structure is similar as desired.

- `DocSysBasics.rsl` (appendix D.2 page 129) – holds overloading of different types in order to facilitate an easy manipulation of the system entities. The file also contains helper functions. This specification is also the basis of the domain, hence the basic structure is similar as desired.

- `eDocSysTypes.rsl` (appendix E.1 page 145) – represents the specific types and specific observer functions introduced when the domain was digitized.

- `eDocSysBasics.rsl` (appendix E.2 page 147) – specifies requirements specific helper and overloading functions.
- `eDocSysWF.rsl` (appendix E.3 page 152) – contains the well formed criteria for the system at hand. These include that a document / person can only be in one place at a time. Consult the specification for further understanding of the criteria.
- `eDocSysCmds.rsl` (appendix E.4 page 153) – manifests the commands available to the system user.

The system definition, complying with requirement 6, 9, 10, 25, and 26 and ensuring well-formedness, dictates that,

```
System′ = Places × DocumentID-set × DossierID-set × ExportID-set
System = {| w:System′ • wf_system(w) |}
Place = Directory × Persons × Locations × RecycleBin × Keys
Directory == mk_dir(DirContents × (Index -m-> Directory))
Persons = PersonID -m-> Person
Locations = LocationID -m-> Location
RecycleBin = Document-set
Keys = Key-set
```

The 'Edit' command available complies with requirement 13(A.16d), 16, 19, 29, 35(b) and 37:

```
M: Command → System → System
M(cmd)(places, docids, dosids, copyids) ≡
  case cmd of
    mk_Edit(person, plid, time, document, (te,fe)) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        let docs = obs_Group(document,docids) in
          assert(hasPermission(person,document,Edit) ∧
          person ∈ rng pers ∧
          mostRecentVersion(document,docs) ∧
          docs ⊂ obs_Documents(person));
```

It is asserted that the person attempting the editing: Has the permission to perform this command on this document, is in fact a member of that place (has a profile), that the edit is performed on the newest version in the document group, and that he has the document in question.

```
            let docid:DocumentID • docid ∉ docids in
              let doc:Document •
                obs_ID(doc) = docid ∧
                obs_Time(doc) = time ∧
                obs_Contents(doc) = te(obs_Contents(document)) ∧
                obs_Type(doc) = version ∧
                obs_Creator(doc) = obs_ID(person) ∧
                obs_PlaceID(doc) = plid ∧
                obs_Ancestor(doc) = mk_did(obs_ID(document)) ∧
                obs_Signatures(doc) = {} ∧
                obs_DirMembership(doc) = obs_DirMembership(document) ∧
                obs_DossierMembership(doc) = obs_DossierMembership(document) ∧
                obs_CommandLocks(doc) = obs_CommandLocks(document) ∧
                obs_Events(doc) = obs_Events(document)
              in
```

When editing a document a new 'document' is created with almost the same attributes – contents is new and signatures are no longer valid and are therefore cleared.

```
            let evt:Event •
              evt_type(evt) = Edit ∧
              evt_executedby(evt) = obs_ID(person) ∧
              evt_time(evt) = time ∧
              evt_place(evt) = plid
            in
```

The edit event is logged.

```
              (places † [plid ↦
                (dir, pers † [obs_ID(person) ↦
                  ((person \ {document}) ∪
                                     {addEvent(document,evt)}) ∪
                      {addEvent(doc,evt)}],
               locs,bin,keys)], docids ∪ {docid}, dosids, copyids)
end end end end end,
```

The system is updated with the new document by inserting the revised document in the person.

5.7 Glossary

Administrators maintain user profiles, user *permissions* and *directory* structure. They are the electronic version of the domain's Local Administration.

Ancestor This describes the identification of the *document* from where a given *document* is copied.

Command can be one of the following: CreateDoc, CreateDos, Copy, Edit, RemoveDoc, RemoveDos, GetDocFromDos, PutDocInDos, GetDosFromDos, PutDosInDos, GetDocFromDir, PutDocInDir, GetDosFromDir, PutDosInDir, ExportDoc, SignDocument, ResetDocMembership, ResetDosMembership, ReturnDoc, ReturnDos, SendDoc, SendDos, SetDocPermission, SetDosPermission or Merge. In other words, it is the possible manipulation that can be done on a *document* or *dossier* via the system.

Copy *Document Type* choice – describes that the *document* is a copy that has not yet been edited.

Directory A *directory* is a hierarchy of named *indexes*. The hierarchy structure is maintained by *administrators*, i.e. they create and delete the *indexes*.

Document Represents a generic electronic document with an undefined contents. The attributes unique identifier, time of creation, *document type*, creator, *place* of origin, *signatures*, *membership*, *ancestor*, and contents are always available for observation.

Document Group is a collection of *documents*. They are combined so that all *versions* of a given *copy* or *master* are grouped together with the latter, i.e. no *document* can exist outside a document group, which always has a cardinality of one or more – the one being a *master* or *copy* and the rest *versions* this.

Document History is a chronological history of the *document group* with relevant information such as creator and time.

Document Type The document type is always equivalent with one of the following three notions: *master*, *copy* or *version*.

Dossier A dossier is a container of other dossiers and *documents*, i.e. it is a way of structuring *documents*.

Event History is a chronological history of performed *commands* on a given *document* with relevant information such as time and performer.

Export is a physical manifestation of the digital document. It is manifested on paper via printer or other kinds of media when performing the 'export' *command*. The *command* is logged, but confidentiality of the export is supposed to be upheld by the the user performing the export.

Foreign system is a computer system of unknown configuration. It either wants to retrieve information from the document system or the latter seeks information in the foreign system.

Ghost A reference to a *document* or *dossier* placed in the container where the *document* or *dossier* has *membership*. It also contains the id of the *person* who is currently in possession of the object.

Index The smallest stationary unit in a *directory*. It contains zero or more *documents*, *dossiers*, and other indexes. An *administrator* can equip an *index* with zero or more locks preventing access for *persons* without proper *keys*.

Location describes a well-defined physical entity – the only place where exports can be sent.

Key Keys provide access to locked *directory indexes* providing that it is the correct key(s), i.e. different keys open different locks. They can also be used to lock *commands* on a given document. Keys are distributed and suspended by *administrators* who possess every key to every lock at that *place*.

Master *Document Type* choice – describes that the *document* is a master that has not yet been edited – all newly created *documents* are masters.

Membership The membership reflects if there is a particular *directory index* or *dossier* at a particular *place* in which the *document* or *dossier* belongs.

Permissions The ability to perform any *command* on a *document* can be locked with a *key*. A permission equals the ability to perform a certain *command* on a certain *document*, i.e. the virtual *person* of the *user* possesses the required *key*.

Person is a virtual alter ego of the *user* existing only in the computer system. It reflects the *commands* performed by the *user* and describes which *documents* and *dossiers* are currently owned by the particular *user*.

Place is a virtual concept that combines one *directory*, one or more *locations*, zero or more *persons*, a *recycle bin* and zero or more *keys*.

Recycle Bin is where *documents* and *dossiers* go when removed.

Signature Represents the legally binding signature of a given *user*.

User is the one manipulating *documents* and *dossiers* using the EDMS. He performs the manipulation by associating himself with his unique virtual *person* within the system who possesses his *documents* and performs document *commands*.

Version *Document Type* choice – Describes that the *master* or *copy document* is edited.

Chapter 6

Design

Based on the requirement prescriptions derived in the former chapter and the discussion of technologies during the introduction, a distributed system architecture is designed. The aim of this particular design is to create a flexible future-proof architecture.

To achieve flexibility the architecture should not rely on technology instantiations from specific vendors. Rather the design should abstract, through generic interfaces, the distributed system technologies to a level where only the overall principles of the technologies are used. This will result in generic database, communication, and data encapsulation interfaces, which can be made specific during implementation.

The design will be presented in its final form starting with the fundamental principles of the domain model, then moving towards an object oriented design and database structure, and finishing off with information distribution principles and user interface design considerations. The design decisions, which had to be made during this phase, are listed and substantiated in the subsequent chapter and will not be elaborated during the presentation of this final design.

6.1 Basic Architecture

The architecture is intended to be a client/server configuration. The clients offer a graphical user interface (GUI), which enables the users to manipulate the document system. The GUI establishes a link to the server, which contains the entire EDMS, i.e. the client holds no data at all, but is merely a visualization of the server information (shown in figure 6.1). As the figure suggests the user establishes a link to the server – the place – via the client. This link is bound to the user-associated profile in the system – the virtual person. The client then illustrates the state of the virtual person and offers the commands that can be performed. The place structure adheres the requirements, of course, but as hinted by figure 6.1, the documents, dossiers, and keys data remain stationary within the place (at the same position in the server database, for instance). When document containers (persons, dossiers, locations, recycle bin, and directory) possess a document, dossier, or key it means they own a reference to it – the actual data remains stationary and is not moved around within the place. This minimizes data transfers.

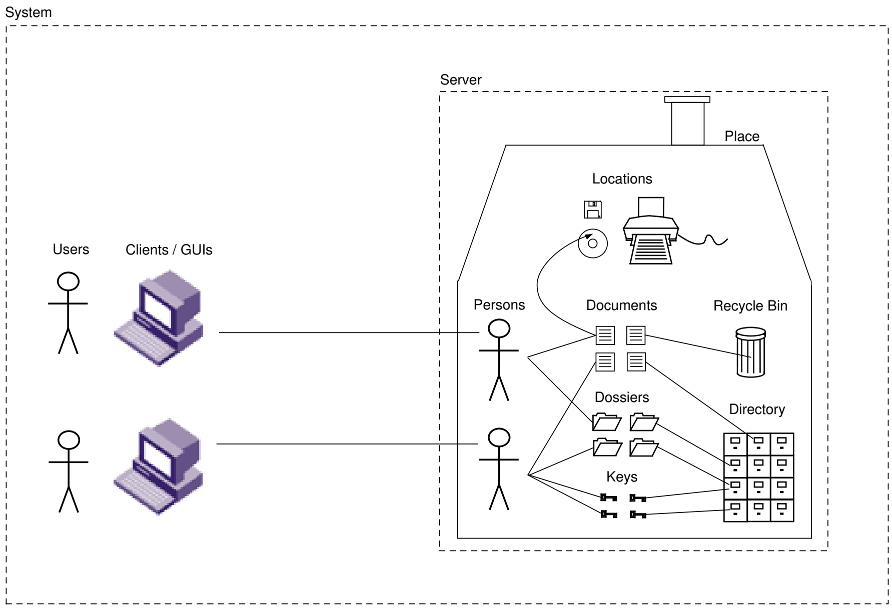


Figure 6.1: Basic System Architecture

The following RSL specification is a more detailed model of the entire architecture – an outline. It incorporates the possibility of having multiple places connected (distribution) via a centralized server (mirror). It also introduces the individual system components.

```
scheme Outline =
  class
    value
      System() ≡ Client[1..a]
               ‖ ( Server ⌈⌉ (Server[1..b] ‖ Mirror) ),

      Client() ≡ ClientConnection,
      ClientConnection() ≡ ClientBusinessLogic
                         ⌈⌉ ClientAdminLogic
                         ⌈⌉ ClientForeignLogic,

      Server() ≡ ‖ ServerConnection[1..d],
      ServerConnection() ≡ ServerBusinessLogic
                         ⌈⌉ ServerAdminLogic
                         ⌈⌉ ServerForeignLogic
                         ⌈⌉ ServerMirrorLogic,

      Mirror() ≡ ‖ MirrorConnection[1..e],
      MirrorConnection() ≡ MirrorAdminLogic
                         ⌈⌉ MirrorForeignLogic
                         ⌈⌉ MirrorPlaceLogic,
  end
```

The principles combined with information flow are illustrated in figure 6.2. The system encompasses:

- one or more **servers** handling a number of connections each capable of containing one of four different kinds of logic layers, e.g. a logic specifically designed to handle administrative commands. Each of these four layers contain the logics of a `Place` and a set of `Commands`, which were outlined during the requirements development. Furthermore, the layers contain replaceable facilities for database communication (`DBLayer`) and network communication (`ComLayer`), both of which will be elaborated later.

- an optional **mirror** that serves as a centralized unit in a distributed server environment. It relays connections from server to server (i.e. place to place). The subject of distribution is addressed later in this chapter.

- zero or more **clients** of three different types

  1. **business clients** which are used during normal document operations such as manipulation of documents and dossiers. They access the `ServerBusinessLogic` of the local server, which, if the system is configured to being distributed, accesses the mirror. More on that subject later.
  2. **administration clients** which access the `ServerAdminLogic` of the local server allowing the administrator to perform special commands.
  3. **foreign clients** which are of unknown configuration. This category represents all third-party software (e.g. middleware), which might require access to some of the data in the system. Because of this they access the server or mirror with special privileges in the `ServerForeignLogic`.

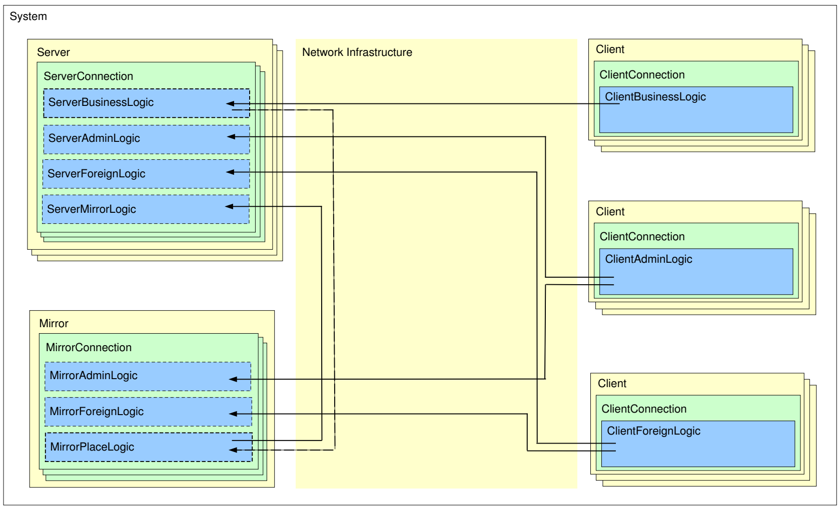


Figure 6.2: Basic Architectural Model and Information Flow

6.2 Object Oriented Design

The design is taken to the next level by outlining the architecture in UML class diagrams. This structure is the product of visualizing the system needs based upon the requirements and the above mentioned basic architecture – in this case the focus will be limited to the place logic. Figure 6.3 depicts the objects aggregation required in order to realize the object `commands`, that holds all commands available to the user in the system. The `place` object is essentially considered to be a database wrapper used by the commands to manipulate the system state in accordance with the domain terminology. The objects aggregated by the place represent the individual entities of a place, such as persons, keys, etc., and are intended to be wrapper classes to their specific part of the database structure, i.e. the classes offer a direct access to all documents, dossiers, persons, keys, locations, keys, recycle bin and directory from where tables of contents (TOCs) can be generated among other things. Database access is achieved via the abstract interface class `DBLayer` that is available to all classes at all times. Manipulating a single entity or data object such as a person or a document is

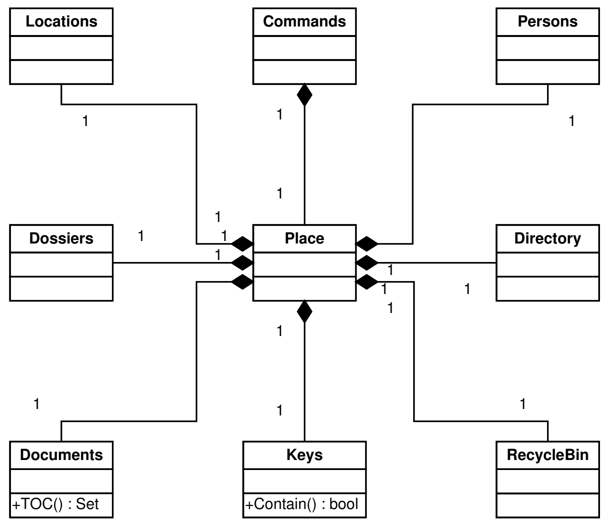


Figure 6.3: Command Object Composition

facilitated through the classes in figure 6.4-6.8. The layout is a direct interpretation of the domain entities and their relationships. Select attributes and methods are displayed in the diagrams to provide a general idea of what the full class diagrams would look like. As mentioned in the introduction we will not present the full object oriented design – although it has been carried out in rough sketches. Instead a few of the main class diagrams will be presented to illustrate the overall object oriented design.

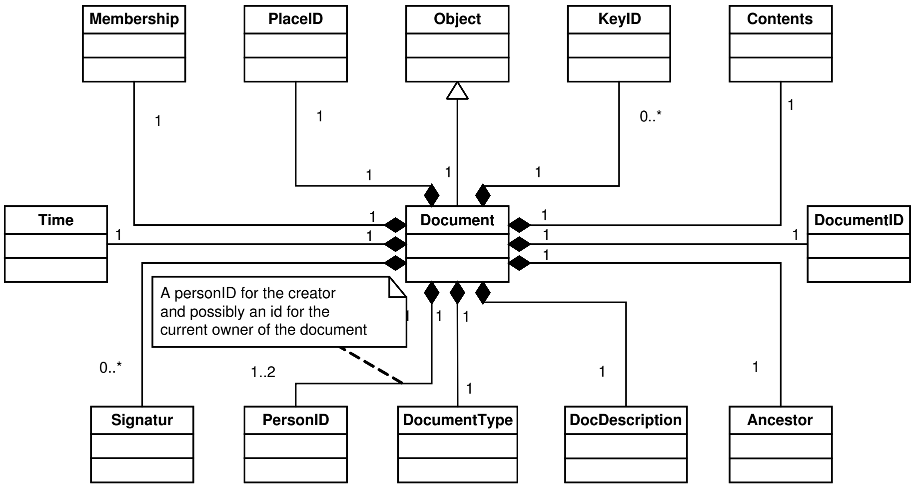


Figure 6.4: Document Object Composition

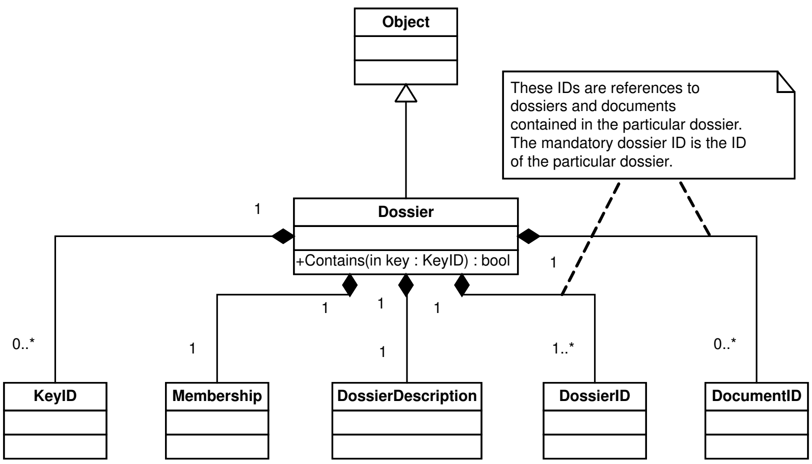


Figure 6.5: Dossier Object Ccomposition

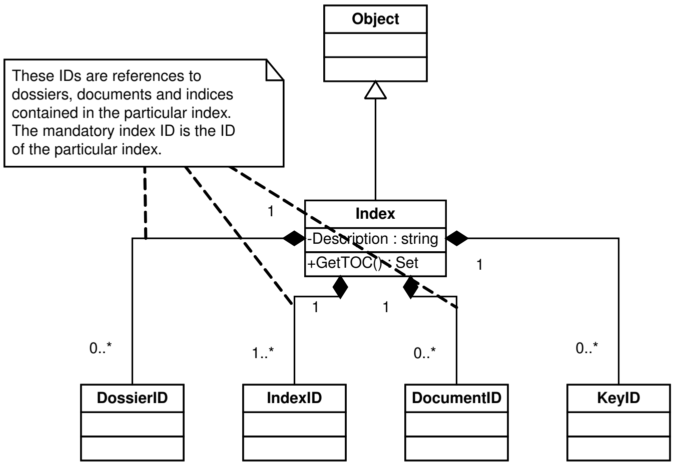


Figure 6.6: Index Object Composition

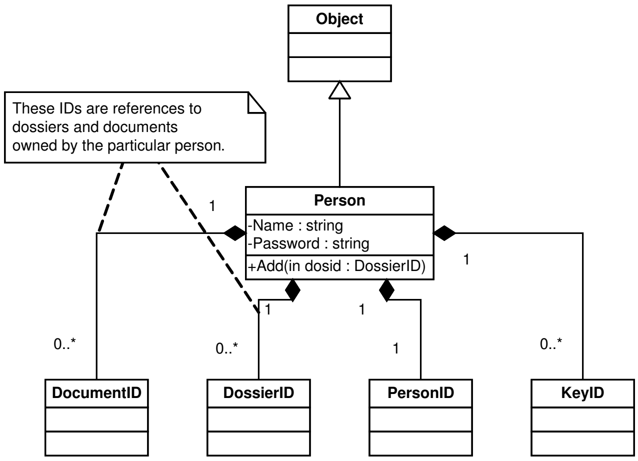


Figure 6.7: Person Object Composition

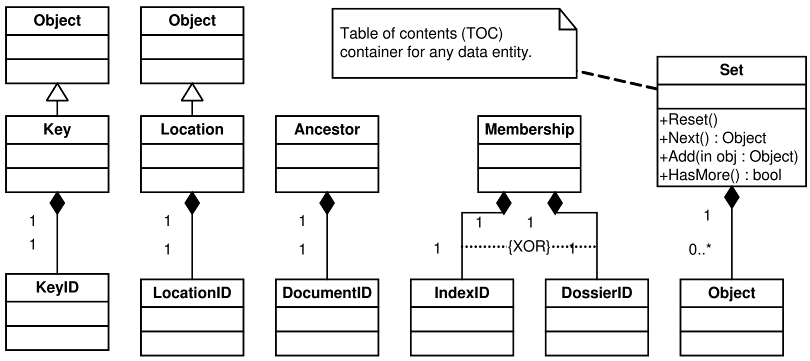


Figure 6.8: Miscellaneous Objects Composition

6.3 Database Design

As previously hinted, all system generated data is stored in a database, e.g. RDBMS, memory or files. The structure of this data can be derived from the object oriented design and the requirements, which is shown in figure 6.9. The figure depicts a number of containers that holds zero or more data structures – each structure contains the fields described in the particular container. The figure suggest names for the containers and their fields that refers to the requirement prescriptions. The fields can be deducted by analyzing the attributes of the objects in the OO design above. The **emphasized** fields represent a unique identification (primary key) for the particular data structure. The arrows constitute an entity relationship, i.e. how the containers relate to each other. It is now inherent that the system calls for a predetermined number of ways to access the database. Adopting the example methods described in the OO design results in the following signatures for the selected database functions:

```
variable
  db : Database

value
  -- Does a specific key exist ?
  KeysContain : KeyID → read db Bool

  -- Retrieve a list of _all document groups
  DocumentsGetTOC: Unit → read db Document*,

  -- Does a specific dossier contain another specific dossier ?
  DossierContains: DossierID × DossierID → read db Bool

  -- Retrieve a TOC of a directory index
  IndexGetTOC: IndexID → read db Document-set × Dossier-set × Index-set

  -- Add a dossier to a person
  PersonAdd: PersonID × DossierID → write db Unit
```

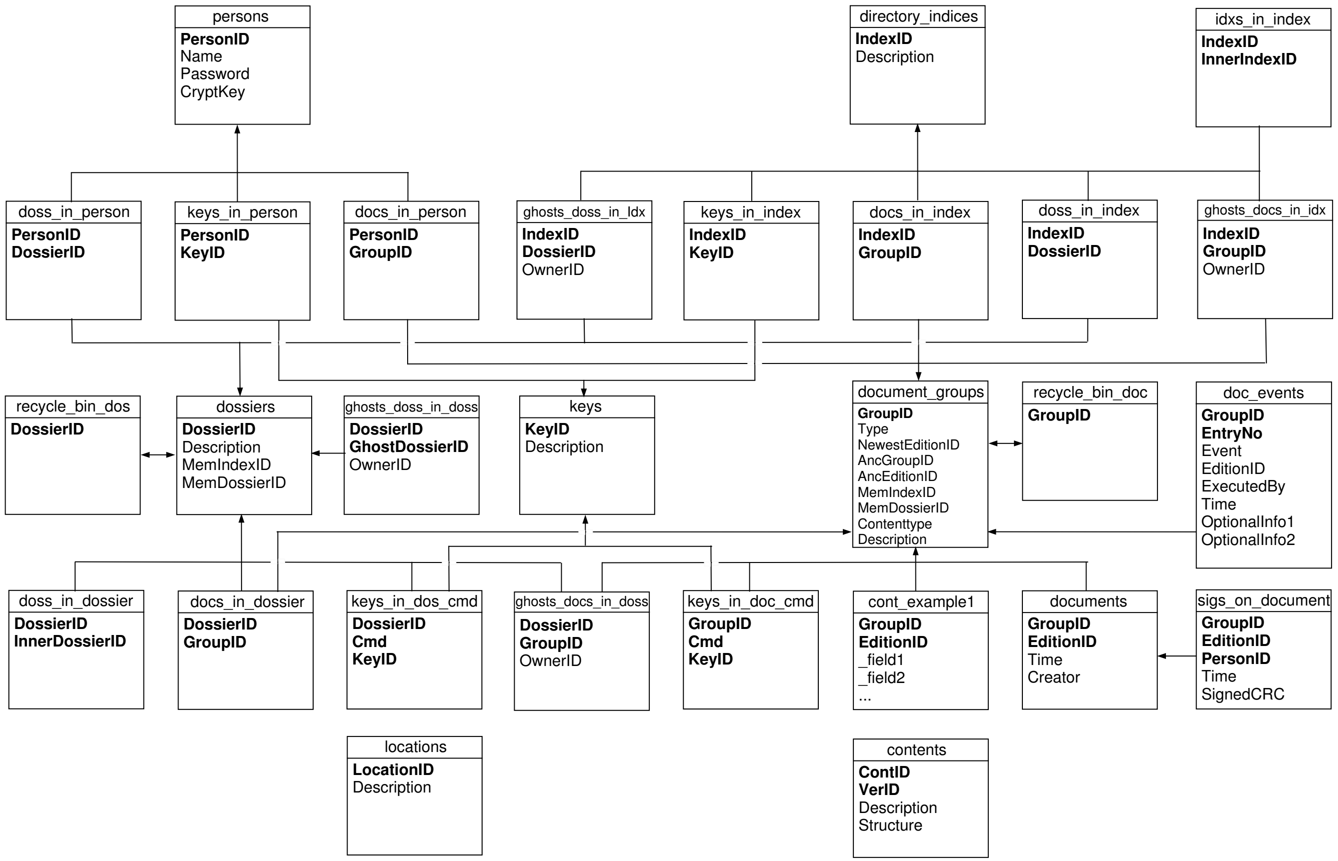


Figure 6.9: Database Structure

6.4 Contents Management

The contents of a document has been handled as an abstract single entity in the domain model. This has to change in order to store it digitally in an efficient manner. When using digitized contents one might want to perform certain actions based on a specific part of the contents. If it is treated as a single entity by the system these actions become exceedingly difficult – if not impossible – to implement. The solution calls for contents to be split up into sub-categories stored individually.

This means that document contents has to be defined in detail for each type of required document. At a later date one might want to modify these definitions, e.g. subdivide parts of the contents even further. To allow for easy contents definitions and modifications it has been attempted to make the contents management system as flexible as possible by allowing the user to add new contents types on the fly.

This is accomplished by introducing a centralized structure specification for each type of contents as well as version control of these specifications. Each document is to be tagged with information about which type and version of contents they make use of. The corresponding contents specification is then used for storing and retrieving the different parts of information from the database and then putting them together to form the overall contents. This results in flexible contents management due to the fact that

- there is no need for a system core updating when introducing new document types,
- a specialized administrators tool can aid in the creation of contents type definitions by providing a graphical user interface which also manages the allocation of space in a database for the new contents type,
- versioning the contents specifications introduces the possibility of modifying existing types of contents, e.g. if a part of the contents has to be discretized further.

A concrete example of this rather abstract type of contents management is shown in the subsequent implementation chapter, where XML schema is used as a contents structure specification language in conjunction with a customized XML mapping layer on top of a relational database.

6.5 Distributed System Architecture

Meeting the demands of being able to connect to the system from a client application and exchange documents across different places calls for a distributed architecture at the core of the system. The design detailed at the moment can be considered a traditional 3-tier client-server architecture consisting of a client, a business logic server (the place), and a database for storing information. This is now expanded to a distributed n-tier architecture, and the data exchange and security aspects of this design will be described in the following.

6.5.1 Data Distribution Between Servers

Documents, dossiers, and directory indexes are shared between places using what will be referred to as the 'mirror' concept. This scheme allows for local references to remote places instead of references always pointing to documents, dossiers, and keys stored locally. In other words, a dossier at place A might contain a reference to a document stored at place B, indicating that the dossier contains a remote document. This means that although some document or dossier physically exists at a given place there might not be a reference to it at that place because it has been assigned to a container at some other place. Potentially, it allows for a person to possess documents, dossiers, and keys originating and stored at other places.
Four situations exist where communication needs to go through the mirror . . .

1. . . . when a command is to be performed on a document or dossier and some of the involved elements (document, dossier, keys, index) are stored at a remote place (information push).

2. . . . when a table of contents of a locally stored dossier, index or person is being created and some of the elements in the table are stored at a remote place, then the descriptions of the elements are requested through the mirror (information pull).

3. . . . when a table of contents of a remotely stored dossier, index or person is to be created then the place requests a table of contents from the mirror which forwards it to the place where the container is stored. If parts of the table contains remote references in relation to that place then the principle of item 2 is used (information pull).

4. . . . requesting a table of contents of the root of the directory is a special case as this requires the mirror to ask all places for a table of contents, like item 3, of their directory root (information pull).

Distributed Flow of 'Push' Information

Figure 6.10 illustrates the 'push' information flow from the client to its local place and, if necessary, from there to other places through the mirror. The individual steps on the figure are numbered and explained in the following.

1. A client sends a message to the server it is connected to locally, e.g. a request that a command is to be carried out on a specific document or dossier.

2. If all involved elements of the command (document, dossier, index, etc.) are stored locally then the command is processed here and the information flow ends. Otherwise the local place forwards the message to the mirror.

3. The message is processed by the mirror which involves:

   (a) checking that the relevant document or dossier is possessed by the person or is available at a specified directory index from which he is requesting it.

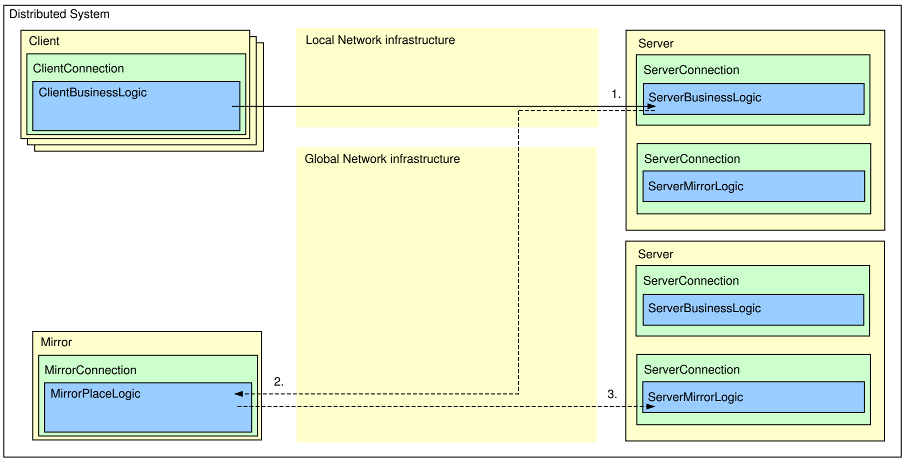


Figure 6.10: Information Flow: Distribution – Information Push

(b) checking that the person possesses the required keys for the command.

(c) updating the involved places to reflect that the command has been executed.

The mirror itself does not contain any information, instead whenever it needs to carry out (a), (b), and (c) it collects the necessary information from the places involved in the operation. When it has decided that the given person is allowed to perform the command it updates the involved places through their mirror logic to which it is connected. This mirror logic allows for direct modification of the state of the place.

Distributed Flow of 'Pull' Information

Figure 6.11 illustrates the 'pull' information flow from the collaborating places to a client connected locally to one of them, e.g. when a table of contents is being put together for a specific directory index which contains documents or dossiers spread across remote places.

1. A client requests a table of contents of some container (his person, a dossier, or a directory index).

2. The local place checks if the container is stored locally, if so the table of contents is generated locally and if it contains remote elements their descriptions are requested from the mirror. If the container is stored remotely the local place asks the mirror for a table of contents of the container.

3. If the mirror is asked for a description of an element it retrieves the description from the place where the element is stored and returns it. If the mirror is asked for a complete table of contents of a container it redirects the request to the place where the container is stored.

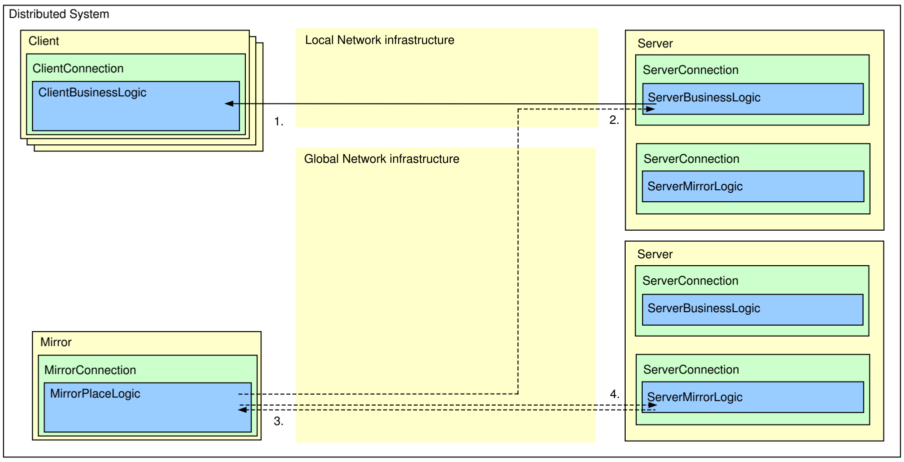


Figure 6.11: Information Flow: Distribution – Information Pull

4. If a remote place is asked by the mirror for a table of contents and this table contains elements remote to this place, the remote place asks the mirror for descriptions of these.

A detailed RSL specification of the information flow in this distributed system architecture can be found in appendixes G.1-G.24 starting page 172.

6.5.2 Secure Communication Layer

Communication between tiers is managed by the communication layer which is an abstraction of secure data exchange across some, yet to be specified, line of communication. This layer provides basic client server communication functionality as well as methods needed for authentication and secure tunneling between two parties. The cryptographic principles required for authentication and tunneling have been introduced earlier during the technology studies and they are elaborated in appendices B.1 and B.2 page 120. The communication layer is

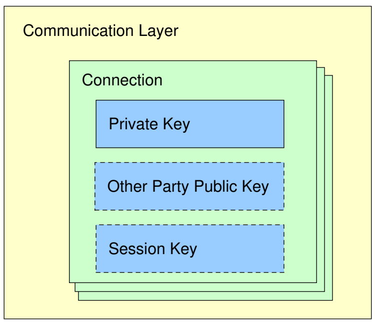


Figure 6.12: Communication Layer

intended to be used by both the client, the places, and the mirror. Its structure is outlined in figure 6.12 and the methods made available by each component of the structure are described in the following (they are highlighted):

`Communication Layer`: This provides the basic functionality required to establish a `Connection` between two parties. One party can choose to **accept** incoming connection attempts while the other party tries to **connect** to it. The outcome of this is a `Connection`.

`Connection`: This represents an established line of communication between two parties. Through this they are able to **send** and **receive** information. Furthermore, they can **authenticate** each other and establish a secure **tunnel** using cryptography.

`Keys`: Three different encryption keys are handled by a `connection`. What they have in common is that they can be either symmetrical or asymmetrical and they are all able to **encrypt** or **decrypt** a given text. The `Private Key` is used for authenticating yourself to the party at the other end of the connection. The `Other Party Public Key` is used for authenticating the party on the other end of a connection. Furthermore, the `Private Key` and `Other Party Public Key` provide means to **sign** data and **verify** signatures, respectively. A `Session Key` is created when a secure tunnel is to be used for the connection.

The protocol of how the methods are used when two parties are connecting, authenticating, and establishing a secure tunnel between each other is specified in detail in appendix F.1 page 167.

6.6 User Interface Design

The user interface to the system is heavily dependent on the business logic and document contents of the specific domain in which it is instantiated. Therefore, development of the user interface is not feasible at this stage. Instead, this is left as one of the main focus areas when developing a domain specific system on top of the platform provided by the EDMS.

Chapter 7

Design Considerations

During the design phase several decisions had to be made, such as choice of architecture and how to adopt and where to insert the domain model in the greater picture. This chapter presents the central design choices along with their alternatives.

7.1 Client/Server vs. Web-based

The primary design choice, effecting the rest of the core architecture, is the decision to create a client/server solution as opposed to a web based solution. These days web based clients are commonly regarded as the most future proof and easily maintained system solutions. Updates are centralized and automatically distributed to clients, and the Internet and web browser technologies are here to stay, ensuring that the clients will be compatible with future web enabled operating systems.

One of the problems with web clients, at the moment, is that they place restrictions on the design of user interfaces and interaction with the hardware of the machine they are running on. If a user interface is based on the graphical elements provided by standard HTML you are forced to present information using static tables, buttons and text. If one wants to create a convincing digitized replica of a document, like the Microsoft Word editor, a more dynamic environment is required where user interactions have a direct effect on the way information is presented.

It is possible to develop a dynamic graphical user interface using existing web technologies, however, these solutions involve embedding applications within the web browser (e.g. Applets, ActiveX, Flash), which – except for the advantages of easy update distributions – do not differ much from an actual client/server solution. They normally also suffer from severe security limitations with regards to hardware interaction with the client machine. Based on these considerations a client/server architecture has been decided upon, as we believe that this will increase the possibility of designing a usable domain oriented GUI.

7.2 How to Adopt the Domain Model

When designing an EDMS based on the domain model one has to consider carefully how to adopt the terminology and structure of the model in the most fitting manner. The document system has to be a well-defined entity: Where does it begin and what is constituted by it?

- The entire model could be adopted as a single document system. This means that the document system would be divided into places being able to cross communicate. These places could correspond to different departments within the same company, or different companies cooperating using the same instance of the document system.
- Another way would be to consider each place as an independent document system and let the design describe the document system as a single place. Communication between persons within this place would correspond to internal transactions in the document system, while cross communication between places would correspond to transactions from and to external document systems.

The problem with the first approach is that one limits the model to describe only the internal mechanisms of the document system. Nothing is known about the world surrounding the document system and how one should interface to it.

In contrast, the second approach which considers each place as an isolated EDMS has the advantage of having a well-defined context of the EDMS represented by the rest of the model. This makes it easier to consider when and how interactions between document systems should take place and how to centralize control of these transactions.

Based on these considerations, the second approach was decided upon. Having the system placed within a well-defined context aided in the design of the 'mirror' concept presented in the design.

The next decision is how to adopt the Place model in a modern client/server architecture. One possibility would be to let clients correspond to persons and let the server be the directory. This would mean that commands such as 'GetDocFromDir' would physically move a document from the server to the client. Modifications to documents would be carried out locally on the client machine and then handed back to the directory on the server when performing a 'PutDocInDir' command.

Another possibility, which is the one applied in the final design, is to implement the place model completely on the server side and then let users bind themselves to virtual persons through the client software. Using this approach all documents are kept on the server and manipulation of documents are carried out on the server side when a user requests that the person he is bound to performs a certain task. This naturally results in absolute dependency on the server since no work can be done without being connected to it. This dependency aside, by keeping persons as a container on the server several design issues, when dealing with distributed systems, are solved in a simple yet very effective manner. These issues include:

Roaming profiles Whenever the user accesses his person he possesses the same documents and dossiers regardless of which client machine he is accessing the system from.

Minimizing data loss If the connection between the client and the server should suddenly be interrupted then the chance of data loss is minimal since no important data resides on the client machine.

Thin clients Since all actual management of documents takes place on the server the clients are simple visualizations of information requested from the server and all user interaction is redirected to the server for processing. This means that changes in business logic can be maintained centralized, like when using a web based solution.

Semaphore protection Because it is required that the virtual person possesses a document in order to modify it, it is guaranteed that race conditions – simultaneous modification of the same document – will not occur.

7.3 Modular Structure vs. Specific Technologies

When expressing a system design one has the option of choosing a modular architecture based on replaceable components as opposed to a design optimized for specific irreplaceable technologies. It all boils down to a question of performance vs. future proof solutions. If specific technologies are dictated then it is possible in the design to optimize the data flow and utilize specific performance techniques of the chosen technologies. However, in doing so you limit the options of replacing, for instance, the database if better types of databases should surface in the future.

Based on the opinion that future proof solutions outweigh optimized solutions here and now, a less optimized modular design has been chosen. Technologies such as the database, cryptography, network protocols, and data exchange formats have been made replaceable components in the design. For instance, instead of basing the secure network communication design on the SOAP protocol – which is a well-established standard for secure web transactions provided by W3C – network communication has been abstracted to a more academic level consisting of generalized interfaces to the cryptographic functions required for (among others) establishing a SOAP-compatible communication protocol.

7.4 Database Design

In the spirit of modular design, the database structure outlined in the design attempts to isolate different domain model entities (like documents and dossiers) in separate database tables. Another approach could have been to let different types of entities be represented in the same table but distinguished by a type column. This results in fewer but wider database tables which leads to fewer database table joins. This might increase performance when doing database lookups, but on the other hand, it makes attribute extensions of existing entities more of a nuisance since it eventually becomes difficult to manage increasingly wider tables.

As a contrast the modular database design consisting of more tables is easily maintainable and allows for basically unlimited extensions, for instance to the types of document contents without cluttering the overall structure. Ultimately, this is the main reason for the choice of a modular database structure.

7.5 Contents Management

The contents management of the document system has been designed to be as flexible and future proof as possible. This requires a dynamic database structure as opposed to a more traditional static set of tables and fields. The dynamic structure is realized at the price of performance since some degree of contents processing and analysis has to be carried out when interacting with the database. If performance is a priority several other design choices might have been more relevant:

- Contents can be handled as a single value placed in a field in the database. This approach does not allow discretized data mining of contents, and consequently limits the possibilities of basing business processes on contents input in the business logic.
- Contents can be analyzed and divided into elements of information. A database structure consisting of hard coded tables and fields can then be created to match the types of contents required.

Both of these alternatives may offer better performance but at the same time they limit the possibilities of future expansions. We chose to opt for the future proof dynamic contents management model, which explains the design decision presented in the previous chapter.

7.6 Data Distribution Between Servers

For data exchange between two or more distributed systems the design dictates the use of a 'mirror' server. Instead of using message relaying by a third server one might have designed a direct system to system communication service. However, the advantages of introducing a third party between the systems allows for an easier centralized administration of the list of systems who are to cooperate. Furthermore, the individual EDMS only has to worry about establishing and maintaining one secure external connection. These advantages aside, the differences between the message relaying solution and the direct server to server solution are minimal.

Chapter 8

Implementation

A prototype of the EDMS, based on the requirements and design specifications, is implemented using current available technologies. How this is accomplished will briefly be described in the next sections – some parts of the implementation code will be shown to illustrate how it relates to the domain, the requirements, and the design.

8.1 Technologies

In the following sections the specific technologies involved in the implementation will be listed. As mentioned in the previous chapter, the design does not dictate the use of specific technologies when implementing it. Since this implementation is considered a proof-of-concept optimizations are not a priority. Therefore, technologies have been selected based on their accessibility and the amount of work required to use them for the implementation.

8.1.1 Platform and Development Language

Microsoft Windows, running on x86 compatible hardware, has been selected as the target platform for both the clients and server. This is due to the fact that Microsoft Windows is the most widely used operating system at the moment and also the one most familiar to the authors. `MFC` (Microsoft Foundation Classes) [27] in conjunction with `C++` is used as the development language. This results in non-portable code but allows for native graphical user interface (GUI) design in the operating system.

Instead of implementing the architecture from scratch one might have used a distributed system framework such as Microsoft `.NET` or `J2EE` from Sun Microsystems. However, the document system is considered an alternative to these frameworks so from a prototyping point of view it would seem unsuitable to base it on one of these existing solutions. Furthermore, the development methods dictated by these frameworks are, in some aspects, in contrast to the ones dictated by the document system design (e.g. they primarily focus on web based solutions).

8.1.2 Database and Interfacing

The selected database is the open source MySQL database, which is based on the relational model described earlier during chapter 2.3.2. The database runs on its own server and is accessed remotely through the MySQL ODBC driver for Windows. The database layer derived implementation uses the MFC technologies `CDatabase` and `CRecordset` in conjunction with SQL to access the tables in the database through the ODBC driver.

8.1.3 Networking and Security

Networking is realized using TCP/IP socket programming in a derivation of the communication layer. A standardized encryption toolkit, Microsoft CAPICOM, is utilized for implementing security technology. RSA public key and 3DES secret key encryption facilities – required for authentication and tunneling – are provided by CAPICOM through the use of digital certificates. A simple

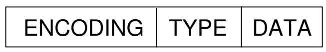


Figure 8.1: Communication Layer Message Format

message format is implemented in the communication layer consisting of three fields shown in figure 8.1. These messages are split, as indicated by figure 8.2, into three network packets when sent over the network using TCP/IP and assembled when received. In short, **ENCODING** indicates whether the next

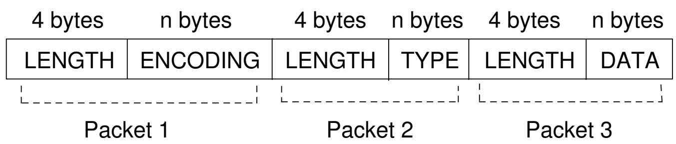


Figure 8.2: Communication Layer Packet Format

two fields are encrypted or unencrypted. **TYPE** indicates the nature of the message, whether it is a request for authentication, tunneling or data. **DATA** is an encrypted nonce for authentication, an encrypted session key for tunneling, or some other data.

8.1.4 Data Exchange Format

XML schemas with (mostly) optional elements are used to generate XML documents, which contain the information to be sent between network tiers. The free XML library MSXML4 by Microsoft is used for schema validation and XML document generating/manipulation. Information requests are represented by XML documents with empty elements. These elements are populated and the XML document is returned as an answer to the request (shown in figure 8.3). This ensures that only the requested information is sent as a response thereby minimizing the bandwidth usage. When extending the schemas with new elements

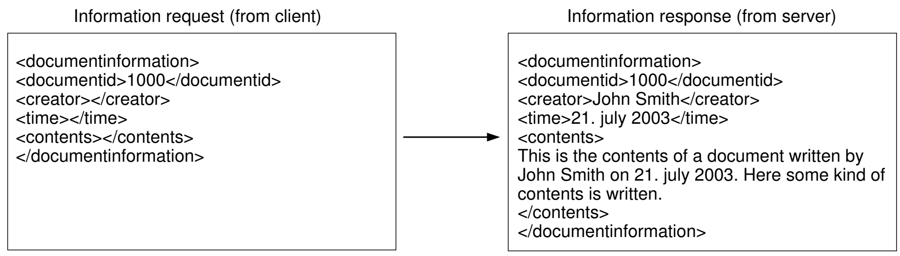


Figure 8.3: Information Request and Response Using XML Documents

(if the system is extended to provide new types of information) backwards compatibility can be ensured by keeping the new elements optional. Furthermore, the use of this format allows for relatively easy two-way interaction with foreign systems such as middleware.

8.1.5 Contents Management

XML is chosen as the language for contents specification due to the already extensive use of it in the implementation. It became evident that XML would fulfill all the contents requirements of the design while being wrapped in a standardized package. XML schemas are used as a contents structure specification language and an XML mapping layer has been implemented on top of the MySQL database for discrete storage of the document contents represented by XML documents. This means that a pseudo XML-enabled database (as described in chapter 2.3.3 page 13) has been created in order to provide dynamic handling of document contents.

8.1.6 Interaction Between Different Document Systems

The concept of the 'mirror' message relay server, used for interaction between systems, has not been implemented in the prototype. How to implement it at a later date has been considered and the current prototype has been implemented with this future extension in mind.

8.2 Examples

To illustrate how the requirements and design are realized in the implementation certain parts of the implementation source code and principles will be shown in this section in order to emphasize the close relationship that exist.

8.2.1 Commands

Since the 'Edit' command, in particular, has been elaborated in the domain- and requirements document it seems appropriate to show how this part of the implementation relates to the original specifications:

```
DSDocumentID DSCommands::Edit(DSPersonID& perid, DSTime& time,
                                DSDocumentID& docid, DSContents& cont)
{
  DSEvent event;
  DSPerson per = DSPerson(m_pDatabase,perid);
  DSDocument doc = DSDocument(m_pDatabase, docid);
  DSDocument edt = DSDocument(m_pDatabase);
```

The needed objects are instantiated and bound to data in the database. It is implicitly checked that the objects exist otherwise an error is thrown (applies to the objects instantiated with an id).

```
  Assert(per.Contains(docid),ERR_PER_DOES_NOT_CONTAIN_DOC);
  Assert(per.Contains(doc.GetKeys(DSEdit)),ERR_PER_DOES_NOT_CONTAIN_CMD_KEY);
  Assert((doc.m_strNewestEditionId == docid.GetEditionID()),ERR_CANNOT_EDIT_OLD_VERSION);
  Assert((doc.m_strContentsType == cont.m_strContentsType &&
          doc.m_strContentsVersion == cont.m_strContentsVersion),
            ERR_CONT_TYPE_OR_VERSION_MISMATCH);
```

It is then asserted that: The document is owned by the user trying to perform the edit, the person has permission to perform the edit on the document, the document to be edited is the newest edition, and the contents type and version match the document group contents type and version.

```
  edt.m_id = doc.NextEditionID();
  edt.m_creator = perid;
  edt.m_type = doc.m_type;
  edt.m_time = time;
  edt.m_ancestor = doc.m_ancestor;
  edt.m_strContentsType = doc.m_strContentsType;
  edt.m_strContentsVersion = doc.m_strContentsVersion;
  edt.m_strDesc = doc.m_strDesc;
  edt.m_membership = doc.m_membership;
  edt.Flush();
  edt.SetContents(cont);
```

The new document edition is created and its attributes are set according to the data provided.

```
  event.m_executedBy = perid;
  event.m_time = time;
  event.m_id = edt.m_id;
  event.strCmd = DSEdit;
  doc.Add(event);
  return edt.m_id;
}
```

An event is set up and logged to reflect the command performed. Finally the ID of the new document edition is returned to the caller. The context of the edit command is available in appendix H.1 page 186.

8.2.2 Contents Management

Figure 8.4 displays the graphical interpretation of a contents specification found in appendix H.2 page 189. As indicated by the figure, expressing a contents type is reduced to creating and connecting boxes as well as dictating the structure origin of data by adding attributes (@) to appropriate nodes. The data origin is described by the parent nodes, i.e. they hold attributes describing the data table name while the name of the children nodes describe the field name in the particular table. The database layer can process an XML schema and return the contents as an XML document.

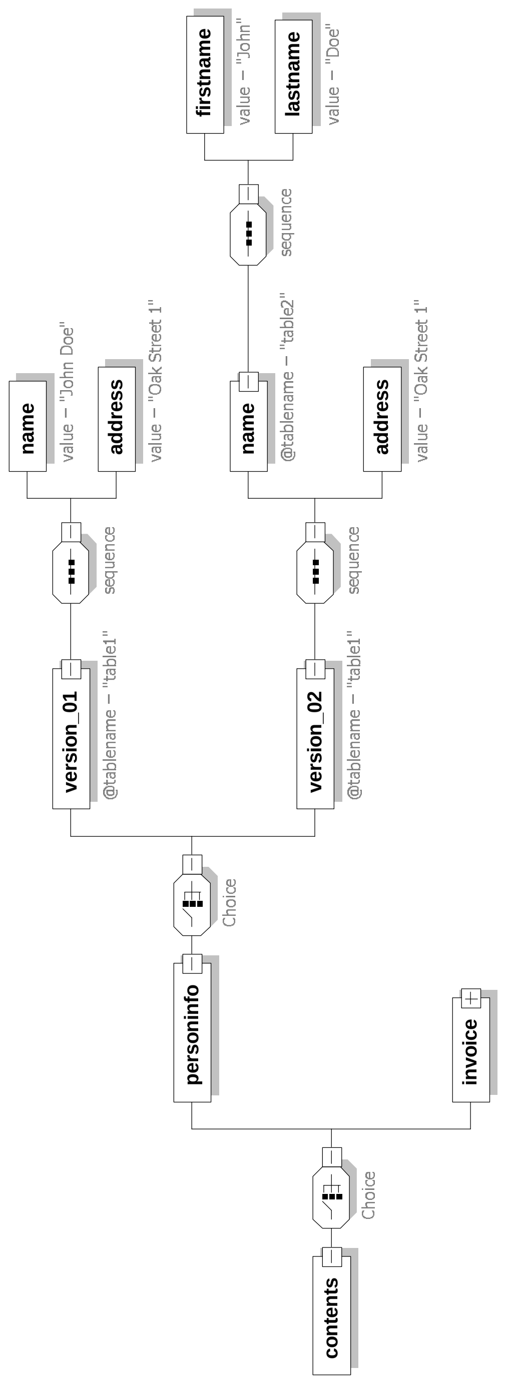


Figure 8.4: Example Layout of a Contents Type Specified in XML Schema

Chapter 9

Prototype Evaluation

Having completed the development steps from domain analysis to implementation it is now possible to analyze the overall result of the domain oriented approach.

9.1 Business Process Building Blocks

The domain analysis of paper document management has resulted in an ontology and a set of commands that can be used to manipulate documents and dossiers within the domain. We believe that these commands fully cover all the basic types of document and dossier manipulations. Consequently, they constitute the fundamental building blocks of paper document oriented business processes.

> **Example:** A meeting between executives takes place in a company where the secretary of one of the executives takes notes. After the meeting the secretary creates a clean copy of the notes, gives the clean copy to her boss for approval and signature. The boss reads the clean copy, signs it, and returns it to the secretary. She hands the signed clean copy over to her intern with instructions to create and deliver a copy to each participant at the meeting and then return the original to her. After this has been done she archives the clean copy in a folder, which is then placed in the company archive.

This business process involves three persons and when they act out the business process using the document management terminology the required building blocks become evident:

> **Susan the Secretary:** I *create* a document containing my notes during the meeting. Later I *edit* my notes into a clean copy which I *send* to my boss.
>
> **Bob the Boss:** I receive a clean copy from my secretary which I read and *sign*. Then I *send* it back to my secretary for distribution to the other meeting participants.
>
> **Susan the Secretary:** After receiving the signed clean copy from my boss I *send* it to my intern for distribution to the other meeting participants.

> **Ian the Intern:** After receiving the signed clean copy from Susan, I make a *copy* for each participant at the meeting. Then I *send* one copy to each of them. Finally, I *send* the original signed clean copy back to Susan.
>
> **Susan the Secretary:** After the clean copy is returned to me by my intern, I archive it. First by *putting it in a dossier*, then by *putting the dossier in our directory.*

Based on this role playing the building blocks can be isolated from the surrounding narrative after which only the document flow and business process building blocks remain (shown in figure 9.1). This demonstrates that a fairly common pa-

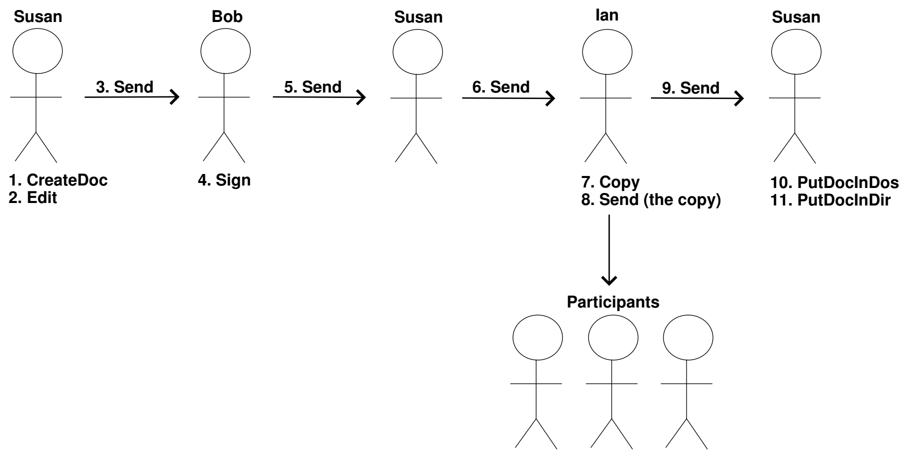


Figure 9.1: Business Process Expressed in Document Commands

per based business process can be expressed in a concise understandable manner using the fundamental building blocks derived in the domain analysis. Because the digital equivalence of each building block has been implemented, the business process can be digitized and kept unaltered in the EDMS.

Once the business processes have been digitized certain advantages of using EDMS become apparent. This may lead to a wish for further business process re-engineering after an instantiated system has been running for a while. In the digital document domain certain tasks – that would take up considerable amounts of time in the paper document domain – can be carried out in seconds by the click of a button.

When studying the previously shown business process it is reasonable to assume that the intern was involved only because the secretary did not have the time to do the copying and distribution herself. Digitally, this task can now be accomplished instantaneously, which effectively eliminates the need for the intern. Based on this digital advantage, it is possible to re-engineer the business process into a different constellation of document commands, illustrated in figure 9.2. The new business process features less cross communication and fewer persons making it overall more efficient than the original.

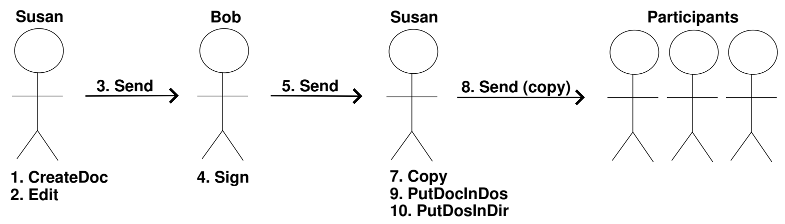


Figure 9.2: Re-Engineered Business Process

9.2 Unexpected Advantages

By choosing to imitate the actual world domain in the EDMS certain advantages have surfaced which were not anticipated when the work began. Most were minor benefits except the ones which appeared in the wake of modelling persons as server side containers of documents. This decision indirectly addressed two major concurrency issues of shared information systems [4].

When several persons are able to access centralized information asynchronously, the situation where they attempt to modify the same information at the same time is likely to occur. This scenario is effectively prevented by the restriction that a document or dossier must be in the possession of the person who wishes to modify it. This restriction indirectly acts as a semaphore protection of documents and dossiers, which prevents conflicting simultaneous modifications from occurring.

Another positive consequence of introducing the server side person container is that it allows for introduction of roaming profiles. That is, the server keeps track of what is in your possession regardless of where you are accessing the system from. This also prevents loss of data if the client machine is stolen/broken as nothing important with regards to the EDMS resides on it.

9.3 Debug Client

In order to ease debugging of the software system a client was programmed exposing all commands and all entities in the system. Screenshots of the software is available in figure 9.3 and 9.3.

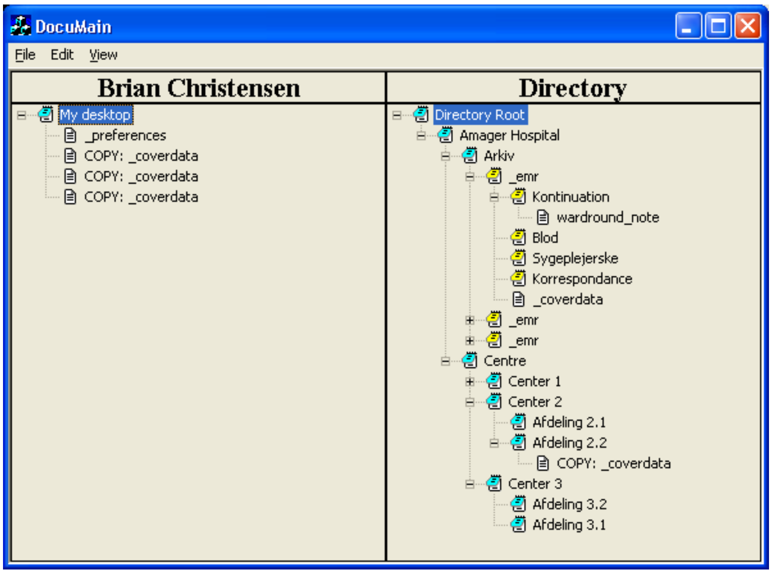


Figure 9.3: Debug Client – Overview

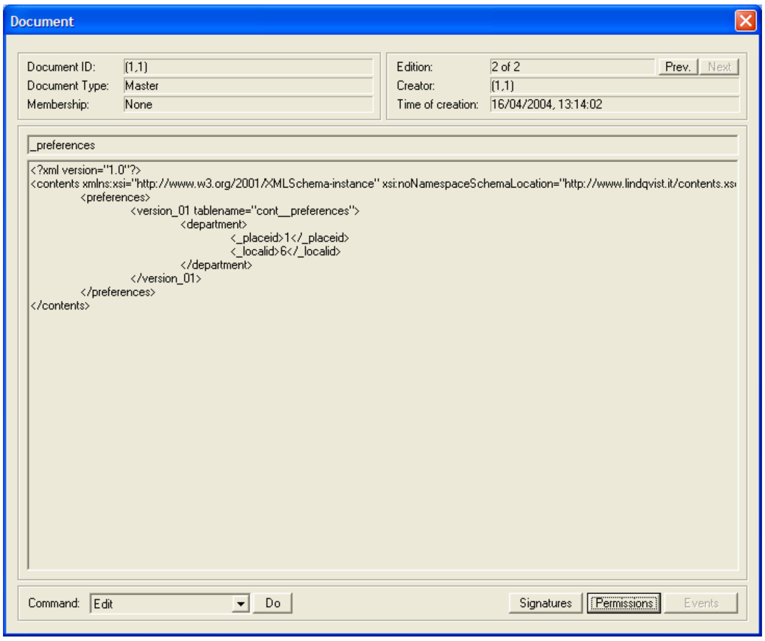


Figure 9.4: Debug Client – Document

Chapter 10

Conclusion

The previous chapters cooperates the initial thesis conjectures by verifying that it is indeed possible to

- create a model of the general paper document domain,
- computerize and extend the model where the digital equivalences are identified and used,
- design and create a general distributed document system based on the computerized model by combining it with existing technologies and software packages.

The domain model resulted in an ontology and a finite number of commands that can be performed on a document in the actual world. The commands constitute a basis of common denominators, which all paper document oriented business processes can be broken into (as demonstrated in chapter 9 page 71). Consequently it is fair to assume that the model can be tailored to support any paper document domain. We will carry out a practical test of this assumption in the subsequent part of the report.

Computerizing the model by finding equivalences between the actual world and the digital world was a relative simple and intuitive process. The model extension was also manageable as it was obvious which features would benefit the user. Furthermore, it was determined what to prevent and support by identifying the undesirable human behavior in the domain development. Again the model resulted in a finite number of commands that, though being extended, appreciated the domain, hence used and respected its terminology. There were unexpected advantages when adopting the domain, the major ones being profile roaming and prevention of conflicting simultaneous document modifications when using virtual 'persons'.

Based on the design considerations and the requirements, the prototype implementation was straight-forward and resulted in an EDMS employing the paper document domain terminology and principles. This provided a platform API consisting of an intuitive scripting language (the document commands) capable of expressing complex business processes in a simple manner. As a consequence of the methodological development approach, the iterations of the implementation process were kept at a minimum.

The original paper domain has been extended through computerization to take advantage of the digital domain. An example of this is logging of executed document commands which automatically provides a detailed event history of every document in the system. Furthermore, the 'keys' concept was adopted and extended to cover individual document commands instead of entire documents. This provided the EDMS with a flexible security layer making it possible to support a wide range of security roles.

Integrating the EDMS core requirements with the technologies of a modern distributed system architecture posed no real challenges – except, of course, minor startup difficulties with regards to the chosen technologies. As planned, all data exchange is encapsulated in XML and placed in a relational database, which implicitly makes it ready for information exchange with foreign systems and future improvements. Transactions and user authentication have been made secure using certified encryption packages in the spirit of general encryption principles.

To summarize, a prototype of an EDMS development platform has been designed and implemented. At the EDMS level it supports document versioning, structuring, and manipulating. At the distributed system level it supports multiple users with individual access rights, secure transactions, data mining, information distribution, future proof data encapsulation and storing. All of these concepts are placed beneath a domain oriented terminology which we believe speaks the language of the end-users.

Part III

MEDICAL RECORD SYSTEM

Chapter 11

Introduction

This part of the Master Thesis carries out a test of the electronic document management system (EDMS) framework presented earlier. It will be based on the development methods outlined by the domain- and requirements descriptions of the world of documents. Furthermore, it will use the implementation of the general document system as a platform for realizing a domain specific document management system.

We have decided to focus on the domain of medical records. As of this writing, there are several ongoing initiatives to attempt to digitize the domain and at the same time shift existing business processes in new directions. During this introduction we will present the current state of electronic medical record (EMR) system development in Denmark, as we see it, and describe our strategy for developing a sub-module of such a system.

11.1 Brief History of Danish EMR

Initial work on nationwide introduction of EMR in Denmark kicked off in 1996 when the Ministry of Health published a report [22] in which an implementation strategy was outlined and development projects were funded. To monitor, assist, and evaluate these and future projects the board 'EPJ-Observatoriet' was established and has published yearly status reports since 1999.

In 1999 a new strategy report [23] was published by the Ministry of Health. This report redefined the concept of EMR to not only include the traditional notes of medical records, but also all other medical information about a given patient (such as medical images, medicine, etc). The report also introduced the idea of re-engineering existing medical record business processes into diagnose-oriented documentation rather than contact-oriented. Contact-oriented is still the most commonly used style of paper based medical record documentation.

Previously, EMRs had simply been unstructured digitized versions of the paper based medical records, also known as 1st generation EMR systems. It was now being suggested to design 2nd generation systems where information was sub-divided into categories and based on diagnose-oriented documentation. Work on a nationwide basic structure and terminology for 2nd generation EMR systems was initiated and eventually resulted in the G-EPJ specification in 2001 [34, 19, 13, 36].

Finally, in 2003 the National IT-strategy for Danish Health Care 2003-2007 [25] was published. This report dictates that all Danish hospitals shall introduce EMRs based on the G-EPJ specification by the end of 2005.

11.2 Current Status of Development

At the moment approximately 13% of all Danish hospitals beds are covered by EMR systems – some of which are 1st generation – and at the current pace of development total coverage by the end of 2005 seems unrealistic [12]. One of the reasons for this delay is that the systems are being developed decentralized and independently of each other by different hospital regions with the only common denominator being the G-EPJ specification. Furthermore, there are different modules within the EMR systems, such as the note and medicine modules, which are being developed in parallel by different companies.

Decentralized development is a good idea as it prevents monopolization, which might stall future improvements and result in high prices. However, because of the decentralization a considerable overhead is generated as it takes a lot of coordination – both politically and practically – before the individual components of the system can be streamlined and pieced together.

To minimize this administrative overhead most of the EMR development projects have decided that there is a need for an integration platform on which the different parts of the EMR systems can be built. Once an integration platform has been decided upon, few doubts remain as to how the different EMR system parts should interface.

At the moment, two major integration platforms are being used in different parts of the country. In Aarhus Amt a completely new integration platform, the Columna Open Architecture [38], has been developed based on the G-EPJ specification. In Hovedstadsregionen's Sygehusfællesskab (H:S) a well-established integration platform, DHE [15] has been decided upon.

The principles behind these are essentially the same. They provide a database structure and interface clearly defining where any specific type of medical information belongs. All data generated by the EMR modules are stored in the database and exchanged via the interface, i.e. the modules can use each others data. Since our main contact has been with H:S the focus will be on their approach to EMR development.

Amager Hospital, which is part of H:S, is currently being used for prototyping modules of the H:S EMR. As mentioned, their approach is to use DHE as integration platform and build individual system modules which interface to this platform. DHE also serves as a bridge to several legacy systems, such as 'Grønt System', which is currently being used in H:S to store general personal information about patients. The goal is to design a single web based portal, which provides seamless access to the different parts of the EMR. A web based client solution has been chosen as it was deemed familiar to the average user while being easy to maintain.

11.3 Our Approach

The main difference between our approach and the current initiatives being taken, is the intention of realizing a future proof 1st generation EMR system, that can be turned into a 2nd generation EMR system eventually. We believe that the 'roll-out' of a completely digitized working environment is difficult enough for the users without introducing new ways of using the medical record. Instead the 2nd generation principles should be introduced later, when the users are familiar with computer interaction.

We intend to use the terms and principles outlined by the fundamental document system to describe and adopt the current domain of medical record management at Danish hospitals. The expectations are that the terminology of the document system will make it easy to describe the document-oriented parts of medical record management in a simple, structured, and understandable fashion. By following this design pattern we also expect the requirements of the 1st generation EMR system to be relatively straight forward and in correspondence with the domain intrinsics.

Finally we expect that most of the implementation time will be spent on developing a client side graphical user interface. Realizing the business processes on the server side should simply be a matter of adopting the business process requirements exactly as written. Once the system has been implemented we will consider how the step towards a 2nd generation EMR system could be taken and briefly evaluate the complexity of this step.

Chapter 12

Domain Development

A domain analysis of the intrinsics and business processes of Danish hospitals is carried out as an extension of the domain model of paper document management presented earlier. To match the scope of this Master Thesis, the domain acquisition has been limited to a subset of a greater truth. Consequently, the domain description will focus only on central aspects of the business processes involving actual management and whereabouts of medical records.

The general issues of document management addressed earlier will be considered solved and extended upon in this domain development. Whenever appropriate, terms and concepts from the document domain are used in order to emphasize that the medical record is a specialization of the document domain. Section 12.12 page 93 holds a glossary which describes specific terms and concepts presented during the domain development.

12.1 Synopsis

The domain development presents a model of generalized medical record management in a Danish hospital. It attempts to describe the behavior of the hospital staff when managing medical records. This includes defining a hospital with a number of centers with departments, outlining the manipulation and structuring of and access to the medical records as well as describing (a subset of) the actual documents.

12.2 Stakeholders

The stakeholders of the domain [35, 37] are listed and described briefly in this section. They are structured in context of the stakeholder structure of the general document management domain. By performing such categorization their direct relationship to the core document management system becomes evident from the beginning.

12.2.1 Global Administration

- **Ministry of the Interior** Maintains a centralized register of all individuals in the country. When a person is born or a foreigner becomes a

Danish citizen he is assigned a social security number by this ministry.

- **The local authorities** Responsible for erecting the buildings when a decision has been made to build a new hospital.

12.2.2 Local Administration

- **Internal services** Manages the locks on doors and filing cabinets within the hospital. Is also responsible for establishing new locations by setting up new desks, shelves, and cabinets.

- **IT department** Is responsible for IT-infrastructure and stability of IT equipment within the hospital.

12.2.3 Person

- **Managing director** Person responsible for a hospital. The highest authority of a hospital.

- **Managing center director** Person responsible for a center of a hospital. The highest authority of a center.

- **Head of department** Person responsible for running a department of a center.

- **Doctor** A person with a master's degree in medicine.

  - **Specialist** A doctor working with a clinical specialty such as a podiatrist, dermatologist, etc.
    - **Consultant** A doctor at a hospital who has a share in the responsibility of running the department in which he is employed.
    - **Resident** A specialist employed in a department at a hospital.
    - **General specialist** A specialist working with a specialty in his own practice.
    - **Scientist** A doctor dedicated to researching. Normally all doctors must research and publish articles, so this particular category overlaps the other categories of doctors.
  - **Junior resident** A younger, not permanently employed, doctor. Part of the specialist education.
  - **General practitioner** A doctor with his own practice. A general practitioner can be consulted without reference. His task is to assess whether a patient requires further treatment elsewhere in the health services, or if he can treat the illness himself, or if nothing further is to be done. The general practitioner will only treat simple illnesses himself.

- **Dentist** A clinician specialized in dental care. Dentists perform preventive dental care and sometimes certain forms of patient treatment, such as examinations, tooth cleaning and filling.

- **Psychologist** A clinician specialized in the human psyche. Within hospitals psychologists treat mental illnesses.

- **Midwife** A clinician specialized in pregnancy and delivery. Their work with women takes place before, during, and after delivery, and their tasks involve pregnancy examinations, delivery preparation, obstetric aid, maternity visits, etc.
- **Nursing staff** Person whose main function is the daily nursing of patients.
  - **Departmental sister** Nurse managing the administrative aspects of nursing services at a clinic or in a department.
  - **Nurse** A person responsible for treatment and medication in accordance with what has been outlined by clinicians. They are also responsible for the daily care and nursing of citizens.
  - **Nursing aide** An educated social worker. Normally responsible for performing simple care and nursing tasks put forward by a nurse.
- **Pharmacist** A person knowledgeable about medicine. Typically he works in the medicine industry or at a pharmacy.
- **Physiotherapist** Works with treatment and recovering of muscles, sinews, and bones of the ill and injured.
- **Medical secretary** Secretary assisting doctors in writing medical records and other routine tasks.
- **Social worker** A person creating social changes through guidance, establishment, and planning of social arrangements. In other words, a person who counsels people with social difficulties or problems.
- **Hospital porter** A person with the responsibility of transporting patients from one place to another inside a hospital.
- **Laboratory technician** A person who works at a hospital laboratory performing probing and analytic/diagnostic work.

12.2.4 Third Party

- **Citizen** Common name for the persons in the domain. Normally this term refers to a Danish citizen by law, but we will expand upon this by letting it include immigrants, illegal immigrants, tourists, etc.
- **Patient** A person currently being treated for an illness somewhere in the health services.

12.3 Stakeholder Subset

To limit the scope of the domain analysis, only a subset of the listed stakeholders, that is, *doctors*, *nurses*, *medical secretaries* and the *IT department*, will be focused upon. These are the primary stakeholders in direct contact with medical records. The rest of the stakeholders have been introduced only to provide an idea of how these four central stakeholders relate to the rest of the hospital organization. Consequently, the other stakeholders will not be elaborated further.

12.4 Interviews

The intrinsics, business processes, and other aspects of the domain have been collected through interviews with a number of people representing each of the stakeholders in the stakeholder subset. The complete domain description has been deducted iteratively by gathering information through conversation with the stakeholders, structuring the information, and sending it back for verification. Furthermore, anonymous medical records from Gentofte Hospital, Amager Hospital, and Herning Hospital have been used to examine the typical layout of contents in medical records.

Name	**Stakeholder**	**Occupation**
Thomas Dalsgaard Clausen	Doctor	Resident, Medical dept., Amager Hospital
Kasper Weibel Nielsen	IT-dept.	IT-architect, IT-dept., Rigshospitalet
Sue Mattoon	IT-dept.	Systems Consultant, IT-dept., Amager Hospital
Merete Lelund	Nurse	Glostrup Amts Sygehus, acute neuro. dep. 28
Mette Andersen	Nurse	Temp Nurse Rigshospitalet, Gentofte
Camilla Christensen	Nurse (student)	Former porter, Odense Hospital, Neuro. dep.

Table 12.1: Interviewed Stakeholders

12.5 Intrinsics

The world of Danish hospitals can be described as a number of hospitals (places) in which there are

- doctors, nurses, patients etc. (persons).
- conference-, patient-, staff rooms, hallways etc. (locations).
- carts with medical records in each department and a central archive etc. (directory).

A medical record can be interpreted as a dossier with a unique identification – a social security number – containing a number of documents and dossiers. The medical record is governed by the same basic rules as general documents regarding manipulation and they will not be repeated here. Instead, the reader is encouraged to consult the domain intrinsics for the general document system in 4.4 page 23.

12.5.1 Contents of Medical Record Documents

Determining and defining the contents of all document types in a hospital is a time consuming process, and it will therefore not be conducted in its entirety in the context of this Master Thesis. Instead one type of document, a medical record note, will be shown as a generalized example of document contents and layout. Medical record notes are created by filling out templates as shown in figure 12.1. To the untrained eye it is difficult to separate preprinting from

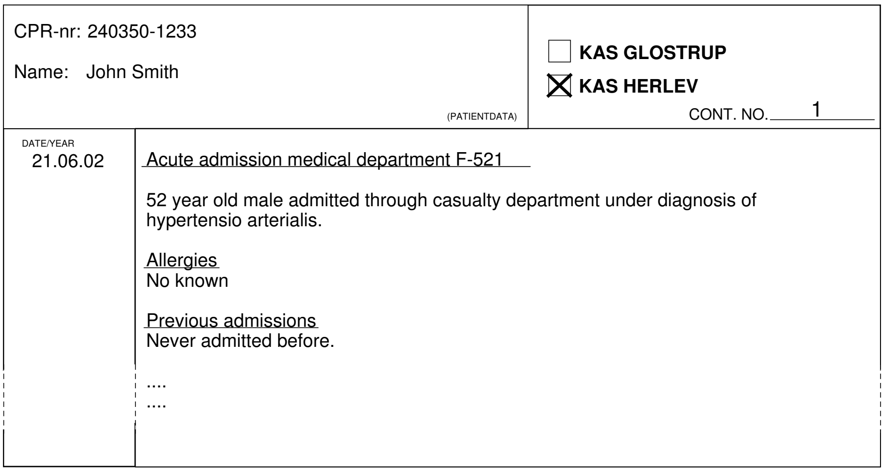


Figure 12.1: Note Page of a Medical Record

information added by the hospital staff. However, it is evident that the note consists of four boxes each with their contents. Through the years a number of general guidelines for the structure of contents in medical record notes have been established. Examples of these can be found in [29] and [11].

12.5.2 The Structure of Medical Records

The structure of a medical record [17] may differ from hospital to hospital, and even between departments within the same hospital. The parameters changing are mostly the title, number, and color of categories inside a medical record. An example of a medical record structure is described in the following:

```
scheme mrlayout =
  class
    type
      Dossier,Color,SocialSecurityNo,
      Name,Address,NextOfKin,BedNo,CAVE,

      DossierDescription == MedRec | Continuation | Blood
                          | Rontgen | NurseJournal | Medicine | _,
      MedRec = SocialSecurityNo × Name × Address
               × NextOfKin × BedNo × CAVE
    value
      obs_Description : Dossier → DossierDescription,
      obs_Dossiers : Dossier → Dossier-set,
      obs_Color : Dossier → Color
    axiom
      ∀ dos:Dossier • obs_Description(dos) = MedRec ⇒ (
      card obs_Dossiers(dos) = 5 ∧
        (∃! dos1:Dossier • dos1 ∈ obs_Dossiers(dos) ∧
                                obs_Description(dos1) = Blood ) ∧
        (∃! dos2:Dossier • dos2 ∈ obs_Dossiers(dos) ∧
                                obs_Description(dos2) = Rontgen ) ∧
        (∃! dos3:Dossier • dos3 ∈ obs_Dossiers(dos) ∧
                                obs_Description(dos3) = NurseJournal ) ∧
```

```
      (∃! dos4:Dossier • dos4 ∈ obs_Dossiers(dos) ∧
                              obs_Description(dos4) = Medicine ) ∧
      (∃! dos5:Dossier • dos5 ∈ obs_Dossiers(dos) ∧
                              obs_Description(dos5) = Continuation )
    )
  axiom
    ∀ dos1,dos2:Dossier • obs_Description(dos1) ≠ MedRec ∧
                          obs_Description(dos2) ≠ MedRec ⇒ (
                            (obs_Description(dos1) = obs_Description(dos2) ⇒
                              obs_Color(dos1) = obs_Color(dos2)) ∧
                            (obs_Description(dos1) ≠ obs_Description(dos2) ⇒
                              obs_Color(dos1) = obs_Color(dos2))
                          )
end
```

The specification dictates certain information on the outer dossier and specific types of dossiers within this outer dossier – colorized categories dividing types of medical information. It is a generalization of the medical record structure, and serves primarily as an example.

This domain analysis will be centered around the continuation dossier of the medical record. This category will be simplified to consisting of four different types of notes: *admission notes*, *ward round notes*, *acute notes*, and *external notes* which are all elaborated in the description of the business processes and in the glossary.

12.6 Business Processes

This section focuses on central aspects of medical records and the problems which surround them. The clinical motivation for creating the medical information and methods of using the information by the hospital staff is left out of the domain analysis.

1. The most widely used type of medical record structure, and therefore also the focus of this Master Thesis, is the chronological ordering of medical notes as opposed to diagnose specific ordering. The pages inside the medical records are placed in chronological order within their respective category. Although a visit to a hospital might involve treatment of several different illnesses they are not documented separately. Instead they appear as an intertwined group of notes in the medical record chronologically sorted based on the date and time the treatments were carried out. This means that a single page of a medical record might contain notes covering the treatment of two or more different illnesses.

2. When a member of the hospital staff needs to access a specific medical record they can search for it in a given department (in the staff room all medical records for the patients admitted to the particular department are stored unsorted) or try to locate it in the centralized archive (sorted by social security number). When the medical record is picked up by someone it is not physically accessible to others, but if correct procedures are followed the whereabouts of the removed record is registered and can be retrieved.

3. When a patient is admitted, his medical record (if it exists) is retrieved from the archives. If time is of the essence or it could not be retrieved

for other reasons, a new medical record is created. The contents of the newly created record is later merged into the archived medical record if such exists. When a patient is admitted, an admission note is created by the doctor and added to the continuation category in the medical record by the secretary.

4. When the medical record for an admitted person is not used by the hospital staff, it is stored in a locked medical record cart in the staff room in the department to which the patient is attached. This prevents immediate access to the confidential medical records.

5. The contents of the medical record should always be available in a single dossier and documents belonging to it should therefore not be removed and placed elsewhere permanently.

6. When a patient is transferred to another department or taken to a medical examination or test the medical record is accompanied. Medical examinations or tests can in special cases be conducted with the absence of the medical record. It is required by the porter and the nurse attached to the patient to make sure that the record is sent along with the patient if necessary.

7. The information generated by examinations and tests such as blood tests or ECGs are added to the medical record when results are available. This could take minutes or weeks depending on the nature of the examination or test. When the information is available it has to be approved and tagged as read by the attending doctor, before it is placed under the correct category. It is up to the doctor who receives and approves the information to place it under the correct category of the medical record. As an alternative, it can be placed as page one in the record in order for the next staff member to see it and, if possible, place the new information correctly.

8. After a patient is X-ray'ed, MR or CT scanned, the pictures are stored in a digital medical image archive with the social security number as reference. This archive is accessible throughout the hospital. The medical record contains for each subcategory a table of contents over the available images as well as descriptions of the images and what medical conclusions could be drawn from them.

9. The key points of the result of any test or examination are added to the medical record under the continuation category as an external note.

10. A medical conference is conducted each morning where medical problems are discussed among several doctors. Matters of dispute or doubt are discussed in order to deduct the correct diagnosis. Medical test results are often presented and diagnosed in this forum.

11. Before ward rounds, a meeting is conducted between a single nurse from a specific department, different doctors, and possibly medical students. During this meeting each medical record of the department is scrutinized. The nurse points out important observations from the nurse journal category which may help the doctor select the most fitting medical approach.

The amount of time spent on these meetings vary greatly between departments and hospitals.

12. During ward rounds the doctor is accompanied by the record cart. He records observations and changes for the particular patient using a dictaphone and occasionally adds information in writing to the medical record.

13. After ward rounds the tapes from the dictaphone and the record cart with all the medical records are sent to the medical secretary. The new information from the tapes and clean copies of the written notes are added to the medical records under the continuation category as a ward round note. Afterwards, the cart is returned to the staff room.

14. When a patient is attended by a doctor without prior agreement, i.e an emergency occurs, the result of the examination and/or treatment is added to the medical record under the continuation category as an acute note.

15. A general information sheet regarding blood pressure, temperatures and fluids is often available at the foot of a patients bed. This sheet is eventually added to the medical record as documentation. Another way often used to maintain this information is for the nursing staff to enter the information directly into the medical record under the nurse journal category – which is reserved for nurse observations.

16. When a patient is discharged the medical record is placed in the archive.

17. The private practitioner of a patient may request information about the treatment of his patient. The hospital staff extracts the essentials from the archived medical record and mails it.

12.7 Supporting Technologies

The medical record domain extends the general document domain with the following supporting technologies:

- Magnetic Resonance (MR) scanner is a way to perform tomography – a non-invasive imaging tool for medical examination purposes.
- Computed Tomography (CT) scanner is a way to perform tomography – a non-invasive imaging tool for medical examination purposes.
- Röntgen (X-Ray) is a more primitive non-invasive imaging tool for medical examination purposes.
- An electrocardiogram (ECG) is a way to monitor the heart cycle of a patient. The resulting graphs are placed in the medical record.
- An electroencephalogram (EEG) is a way to monitor brain activity of a patient. The resulting graphs are placed in the medical record.
- Stethoscope, sphygmo-manometer (blood pressure gauge) and thermometer are used to determine the vitals of a given patient.
- A dictaphone is used to store verbal information when handling patients.

- A light wall is used to display X-ray images if hard copies.
- A projector is used to display X-ray images if digital.

12.8 Management and Organization

Figure 12.2 illustrates the organization and hierarchy of a generic hospital. Some parts of the organization (marked with gray) deals with administration of hospital resources only, while others are involved with actual care of patients (the hospital centres). The latter group of the organization are of most interest with regards to this domain analysis, as they are the ones managing medical records (the IT department is included because of its responsibility in keeping necessary computer systems running). Because of this the administration will be considered a third party stakeholder, and as such will not be elaborated further in terms of business processes and access rights to medical records. A

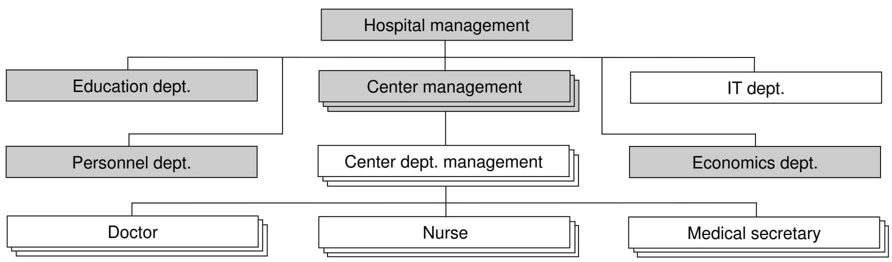


Figure 12.2: Hospital Management Hierarchy

hospital is divided into centers each specializing in particular areas of health treatment, such as heart disease and pregnancy, and each with their own independent administration. A center consists of departments dealing with specific examinations and care within the area of the center, e.g. a pregnancy center would have departments dealing with fertility and gynecology, respectively. Within a department a management coordinates doctors, nurses, and medical secretaries who work alongside in treating patients and manipulating medical records. Each type of employee, however, answers to separate parts of the department management.

12.9 Rules and Regulations

The rules and regulations for the domain of medical records are:

- Security is difficult to enforce, the hospital being a relatively public place. It shall be emphasized, though, that it is desirable to minimize access to the patients, while still allowing next of kin to visit. The hospital has a responsibility of taking care of the well-being of its patients by preventing intruders from disturbing patients.

- The medical records are confidential and should not be accessible to unauthorized personnel. Special laws apply when dealing with personal information in the health care sector. These laws are described in [24].
- The medical record of an admitted patient is stored in the local department. When discharged, the record is stored in a centralized archive for later retrieval if necessary.
- It is required, that the information written in the medical record can be traced back to the author, i.e. all information is tagged with a person id and date.
- It is required by the nurses and doctors to maintain structural information disciplines – all medical information for a given patient is compiled in a single dossier (his medical record) to ensure that all information regarding the patient is kept together.
- During the night all information added to a medical record by a doctor must be drafts in writing (instead of dictaphone recordings). The information will not be added by the secretary before the next day and keeping information in writing during the night will ensure easy access to the information if it is needed during this period.
- Nurses are only allowed to add information to the medical records in the category reserved for them (the nurse journal).
- It is required by the hospital to provide a copy of the medical record upon request from the patient – a minor fee can be charged.
- Access to medical records is not restricted with regards to doctors, nurses, and medical secretaries as they are all subject to confidentiality restrictions. However, since doctors are obligated to produce scientific publications there is sometimes an interest in keeping results of research confidential until publications have been made. Because of this, information might not be shared across departments unless it is vital for the treatment of a specific patient.
- The IT department is not allowed to view the contents of medical records. They are, however, responsible for administration and maintenance of the IT-systems which assist in creating and manipulating medical records – if such exist within the hospital.

12.10 Human Behavior

Extending the human behavior in the domain of medical records yields

- The hospital being a busy place, there is often not time to maintain a well-structured medical record. This can lead to error prone work processes.
- There is a chance of misplacing medical records or a particular document from a medical record as well as placing a piece of information in the wrong one.

- The tables of contents in the medical record listing the medical examinations such as medical images are not updated regularly.

12.11 A Systematic Narrative

Words that are *emphasized* can be found in the glossary page 93:

1. A *hospital* consists of
   (a) an *archive*,
   (b) a number of *centers* consisting of one or more *departments*,
   (c) an *administration*.
2. A *department* consists of
   (a) one or more *staff members*,
   (b) a single *medical record cart*,
   (c) zero or more *patients*,
   (d) one or more patient rooms,
   (e) a single staff room.
3. When a person is admitted to a *hospital* his *medical record* is retrieved from the *archive* if it exists – otherwise a new medical record is created.
4. A *medical record* contains all medical information for a single person generated at the particular *hospital*.
5. A *medical record* is structured according to the specification in section 12.5.2 page 86.
6. The structure of the *medical records* is not altered, nor are complete categories with contents removed from the medical record.
7. A *medical record cart* belonging to *department A* contains the *medical records* of the *patients* of department *A*.
8. The medical record cart is locked and stored in the *department* staff room when not used. It is in use when
   (a) it accompanies the *doctor(s)* and *nurse(s)* during *ward rounds*.
   (b) it is in the hands of the *medical secretary*, who adds new information to the *medical records* dictated by the *doctor*.
9. A single *medical record* can be removed from the cart because
   (a) it is placed in the *archive* (a *patient* is discharged).
   (b) it accompanies a *patient* requiring some kind of medical examination or test.
   (c) the *patient* is transferred to another *department* (the record is placed in a new department cart).

   (d) the *medical secretary* is adding information to it.
   (e) it is temporarily used for reference by a *staff member*.

10. The type of information created by a *doctor* which can be added to a *medical record* consists of
    (a) an *admission note* created when a person is admitted to the *hospital* (category *continuation*).
    (b) a *ward round note* created during *ward rounds* (category *continuation*).
    (c) an *external note* created by *doctors* from other *departments* or a summary of an external examination or test (category *continuation*).
    (d) an *acute note* created when an emergency occurs and is documented (category *continuation*).
    (e) information created externally such as *medical images*, diagrams, test results etc.

12.12 Glossary

Acute note A filled out note template describing a non-scheduled treatment of a *patient*. This could for example be the administering of special medicine to treat an acute condition. The exact time at which the treatment was performed is also written on the note.

Admission note A filled out note template containing the anamnesis and initial observations of a *patient's* condition at time of admission. The anamnesis is composed of individually titled sections describing allergies, former admissions, dispositions, expositions, current state, state of other organs, medicine, addictions, and social status. This information is deducted by interviewing the *patient* or next of kin. At the end of the admission note the *doctor* writes his own objective observations of the current state of the *patient*.

Archive A centralized archive that contains all *medical records* of discharged *patients*. Often placed in the basement of the *hospital*.

Administration An abstract entity dealing with administration, logistics and maintenance.

Category The *medical record* is divided into a number of categories each reserved for a special type of medical information. The names and numbers of categories vary between *departments*. The category *continuation* is assumed always present, however.

CAVE A field in the *medical record* that reflects special things to consider before treatment, such as allergies.

Center A *hospital* consists of a number of centers each with its own *administration* and specialty.

Continuation A collection of *admission notes*, *ward round notes*, *acute notes*, and *external notes* describing the examinations and treatments of a given *patient*. Each note has a page number so that the *continuation* can be read chronologically.

Department A *hospital center* consists of a number of departments each with its own *administration* and specialty.

Doctor A stakeholder with a master's degree in medicine.

External note A filled out note template which is part of the *continuation* of a medical record, but cannot be classified as being either an *admission note*, an *acute note*, or a *ward round note*. These notes may contain the same sections as a *ward round note*. Furthermore, they may contain sections specifically describing the results of an examination carried out on the patient – such as the results of a CT-scan or an X-ray.

Hospital A place where medical services and treatments are provided by *doctors* and *nurses*, consisting of an *archive*, a number of *centers* and an *administration*.

IT department Is responsible for IT-infrastructure and stability of IT equipment within the hospital.

Medical image an image from an MR, CT or X-ray scanner.

Medical record A compilation of medical information about a *patient*. It is divided into *categories*. Each admitted *patient* has a medical record.

Medical record cart A easily transportable cart containing *medical records*.

Medical secretary A secretary assisting *doctors* in writing *medical records* among other routine tasks.

Nurse Stakeholder from the nursing staff, whose main function is the daily nursing of *patients*.

Patient A person currently being treated for an illness somewhere in the health services, e.g. a person admitted to a *hospital*.

Social security number A unique identification of a person. The identification is issued and maintained by government institutions.

Staff member A *doctor*, *nurse* or *medical secretary*.

Ward rounds Are conducted several times a day and involves one or more doctor(s) and a nurse. They check up on all their assigned *patients* and update the plan for treatment as well as the *medical record* based on examinations of the patient's condition.

Ward round note A filled out note template describing the observations and conclusions made by the doctor when examining a specific *patient* on the daily *ward rounds*. It consists of individually titled sections describing the general state of the *patient*, objective observations, biochemical observations, conclusions, and changes to the medicine which is to be administered to the *patient*.

Chapter 13

Requirements Development

Based on the domain analysis the requirements for a note module of an electronic medical record system will be determined. This will be carried out by instantiating the general document system in the domain of medical record management at Danish hospitals. Using the fundamental building blocks of the document system an attempt will be made to mimic existing business processes as much as possible. Section 13.6 page 103 holds a glossary, which describes specific terms and concepts of the requirements being prescribed

13.1 Stakeholders

As with the domain, the stakeholders are listed in the context of the electronic document system to provide a clear view of the connection between the general document domain and the medical record domain.

Administrators – employees in the IT department.

Users – doctors, nurses, medical secretaries.

Maintenance – IT department

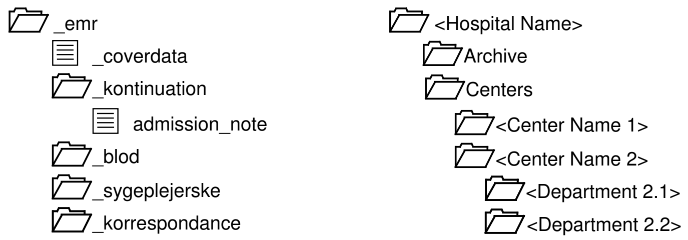
Third party – Patients

Foreign system – Not addressed in this requirements development.

13.2 Business Process Re-Engineering

The following enumerated items are the result of re-engineering the business processes of the domain described earlier in chapter 12.6 page 87. The item numbers refer to the item numbers of the original business processes.

1. Medical records shall be structured as described in the domain analysis. The cover of the medical record shall be represented by a cover page document residing within the outermost dossier, as shown in figure 13.1a. Medical notes shall be placed in chronological order within their respective category dossiers in the medical record.



(a) Medical Record Structure (b) Directory Structure

Figure 13.1: Medical Record and Directory Structure

2. Medical records can only be accessed via a computer. Records shall at all times be accessible in the archive. When access to a medical record is needed a copy of the medical record cover page document is created and placed in the department cart directory index. This copy shall function as a reference to the medical record and correspond to the practice of retrieving medical records for the department cart whenever a patient is residing at specific department. When a medical note is to be edited or created the ancestor of the cover page copy is used to briefly retrieve, modify, and return the full medical record. During this brief time it is read-only to others. The centralized archive is realized as an 'archive' directory index where all medical records are present, see figure 13.1b. All staff members of a given department shall have access permission to this index.

3. When a patient is admitted his social security number is entered into the system. If an existing medical record is available in the archive a reference is retrieved otherwise the user is prompted if he wants to create a new record. If a social security number is not available a medical record tagged 'temporary' is created. When the proper info can be obtained it is entered. If at that time an existing record is available the user is prompted if he wants to merge the contents of 'temporary' record into the existing record, if not then the record remains 'temporary'.

4. A reference (cover page copy) to the medical record for an admitted patient is at all times available in the proper medical cart, i.e. the department directory to which he is attached. The system also enables system users to possess records, i.e. to have a reference to selected records – a kind of 'favorites'.

5. The system prevents invalid medical record structuring and incorrect placement of notes in the records.

6. When a patient is transferred to another department a new medical record reference is made by the destination department in their cart. When the patient is taken to medical examinations or tests, it is required that the examiner has permissions to access and edit the medical record in order to add the examination results.

7. The information generated by examinations and tests are automatically added by the information sender – access permissions shall be present. The information is tagged with category and cannot be placed incorrectly in the medical record. It shall be evident by a table of contents which information needs approval by a doctor.

8. The medical images shall be stored directly in the medical record. A table of contents of the available images are automatically updated. Alternatively the images could be stored in an existing system that can be accessed via links in the table of contents of the medical record – in this case the TOC has to be updated manually.

9. The key points for tests and examinations shall automatically be transferred to the medical record under the continuation category as an external note when external information is added to other parts of the medical record, e.g. medical images or blood tests.

10. During the medical conference all relevant information is accessible according to item 2.

11. During the pre-'ward rounds' meeting all relevant information is accessible according to item 2.

12. During ward rounds all relevant information is accessible through a wireless Tablet PC according to item 2. New information such as dictaphone recordings and notes are recorded directly into the Tablet PC and stored in the medical record.

13. It shall be possible to tag a note as a draft. References to medical records shall reflect the number of draft notes contained in the medical record. It is up the secretary to supervise the references and determine when drafts need clean copying. The information added by the secretary is instantly available to others.

14. When a patient is attended by a doctor without prior agreement, i.e an emergency occurs, the result of the examination and/or treatment is added to the medical record as an acute note.

15. If the information sheet is attached to end of the bed it has to be manually entered into the electronic medical record whenever appropriate. If it is entered into the medical record in the category reserved for nurses it is instantly available.

16. When a patient is discharged the medical record reference in the department cart is removed by a member of the hospital staff.

17. The private practitioner can access all information through the Internet provided permissions are present – no need to rewrite, print and send.

13.2.1 Supporting Technologies

Digitizing the medical records re-engineers supporting technologies with the following:

- A wireless network is needed for supporting wireless components.
- A wireless Tablet PC shall function as the mobile platform for using medical records.
- The dictaphone is replaced by recording speech directly into the medical record through the tablet PC.
- A projector is required to display digital medical images.

13.2.2 Management and Organization

The system incorporates a fine grained permission logic which can prevent access to department directories or even specific medical records and certain types of manipulations on them. It is up to the hospital administration to decide which access policy to follow – open or closed. The former will decrease the potentially fatal situation where a record cannot be retrieved where as the latter protects the information of the medical records across departments.

13.2.3 Rules and Regulations

All rules and regulations are supported by the fundamental document system and will not be elaborated further. It should be mentioned, though, that the ability to provide a copy of the medical record to patients is easy facilitated via the system by granting read-only access to a given patient.

The strict structural discipline followed by the hospital staff regarding medical records can be aided by the system by forcing the user to follow a desired structure.

In the domain, physical keys are required to access some parts of hospital. In the EDMS this is realized by smart cards required by the system at login time. This smart card is the physical key container to the system holding digital keys corresponding to the physical keys of the domain. This smart card also replaces the actual world signature of the user with a digital signature.

13.2.4 Human Behavior

Inappropriate human behavior can be prevented by introducing a business logic which dictates predefined business processes. This includes a strict enforcement of structuring discipline ensuring that medical records always have the correct sub-folders as well as preventing incorrect placement of medical notes.

13.3 Domain Requirements

Performing the domain-to-requirements operation yields well-defined categorized requirement prescriptions, hence the following five sections constitute the complete set of domain requirements.

13.3.1 Projection

It is attempted to adopt as many as possible of the existing business processes without modification in order to produce a user friendly tool. At the same time the less fortunate issues are addressed and prevented in such a way that they do not influence the overall business processes.

1. A.1
2. A.2ab
3. A.3
4. A.4
5. A.5 (extended later)
6. A.6
7. A.7 (extended later)
8. A.9a
9. A.10abcde (extended later)

Projected away

- A.2cde (patients are not system users nor are physical entities such as rooms part of the system)

- A.8ab (the system shall support simultaneous access to a record)

- A.9bcde (the system shall support simultaneous access to a record)

13.3.2 Determinism

10. The documents in the *categories* of the *medical record* shall be sorted chronologically.

13.3.3 Instantiation

11. Roles shall be set up according to the desired security policy. This includes defining access keys to the different directory indexes, protecting individual documents with keys, preventing certain commands to be performed on specific documents without a required key.
12. All users shall be created in the system with an id, name, password and *security certificate*.
13. The *security certificate* shall be available to the client software through a removable *smart card*.
14. The *directory* shall have the structure specified in figure 13.1 page 96.
15. A subset (prescribed in requirements item 9) of medical documents shall be specified according to the concept derived in the document system requirements development – $f(\mathcal{D}) \rightarrow \mathcal{C}$. Figure 13.2 illustrates the principle by showing the contents $\mathcal{C}$ of the previously presented note type and accentuates the relationship between the dynamic data $\mathcal{D}$ (blue) and the stationary template (black). From this relationship the transfer function f can be deducted. A complete specification of the template can be found in J page 195.

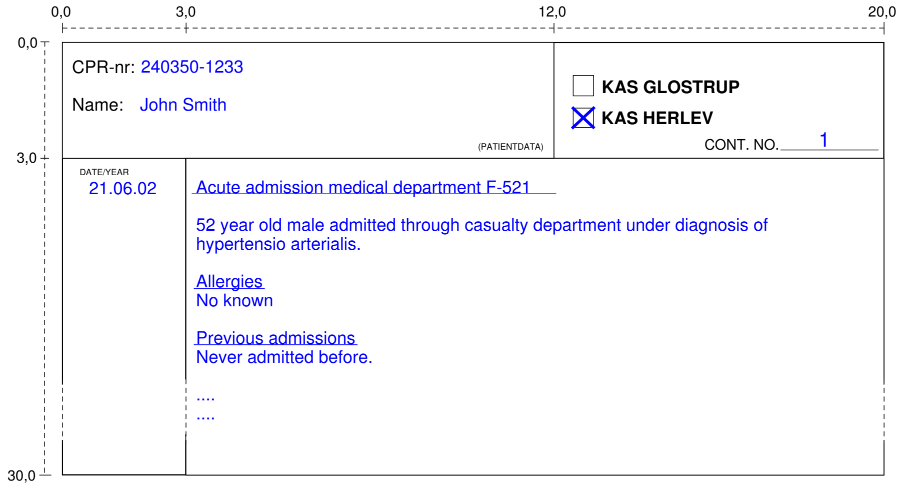


Figure 13.2: Note Page of a Medical Record With Data

13.3.4 Extension

16. It shall not be possible to violate the *medical structure* dictated in the domain development (section 12.5.2 page 86).
17. The structure of the *medical record* shall differ in that the *cover page* (DossierDescription) of the *medical record* will be contained inside the outer record dossier as a document, otherwise the structure shall be identical.
18. Each *person* shall have a *preference document*.
19. All *medical records* shall reside in the *archive*. It shall not be be possible for any user to permanently move the medical records from this index.
20. From a *reference* it shall be possible to access the associated *medical record*.
21. It shall be possible to create a *reference* in either a chosen department *medical record cart* index or on a *person*.
22. Every *medical record* shall at all times be readable to all users provided access permissions are present.
23. When modifications are necessary for a given record the system shall retrieve the record temporarily in accordance with with item 19. This period shall be system dependent and reduced to a minimum. During its absence it is read-only to others in accordance with item 22.
24. None of the fundamental document system commands shall be available directly. Instead a predefined number of *command macros* shall be offered to the user:
    (a) Add a record *reference* to a *medical record cart*.
        - GetDosFromDir (get *medical record* from *archive*)
        - GetDocFromDos (get cover page from *medical record*)
        - Copy (create a *copy* of the cover page)
        - ReturnDoc (return cover page to *medical record*)
        - ReturnDos (return *medical record* to *archive*)

- PutDocInDir (put *copy* of cover page in department cart)

(b) Remove a record *reference* from a *medical record cart*.

- GetDocFromDir (get *copy* of cover page from dept. cart)
- RemoveDoc (remove *copy* of cover page)

(c) Remove a record *reference* from a *person*.

- RemoveDoc (remove copy of coverpage)

(d) Change department and implicitly change *preference document* to reflect new default department.

- Edit (preferences document)

(e) Create a *medical record*.

- CreateDoc (cover page)
- CreateDos (outer *dossier*)
- PutDocInDos (put cover page in outer *dossier*)
- CreateDos (continuation)
- PutDosInDos (put continuation *dossier* in outer *dossier*)
- CreateDos (blood)
- PutDosInDos (put blood *dossier* in outer *dossier*)
- CreateDos (nurse)
- PutDosInDos (put nurse *dossier* in outer *dossier*)
- CreateDos (correspondance)
- PutDosInDos (put correspondance *dossier* in outer *dossier*)
- PutDosInDir (put *medical record* in *archive*)

(f) Add a record *reference* to a *person*.

- GetDosFromDir (get *medical record* from *archive*)
- GetDocFromDos (get cover page from *medical record*)
- Copy (create a *copy* of the cover page)
- ReturnDoc (return cover page to *medical record*)
- ReturnDos (return *medical record* to *archive*)

(g) Create a *note*.

- CreateDoc (create *medical note*)
- GetDosFromDir (get *medical record* from *archive*)
- GetDosFromDos (get *category* dossier from *medical record*)
- PutDocInDos (put medical note in *category dossier*)
- ReturnDos (return *category dossier* to *medical record*)
- GetDocFromDos (get cover page from *medical record*)
- Edit (update number of drafts indicated by cover page)
- ReturnDoc (return cover page to *medical record*)
- ReturnDos (return *medical record* to *archive*)

(h) Save *note* changes.

- GetDosFromDir (get *medical record* from *archive*)
- GetDosFromDos (get *category dossier* from *medical record*)

- GetDocFromDos (get medical note from *category dossier*)
- Edit (edit the medical record)
- ReturnDoc (return the medical note)
- ReturnDos (return *category dossier* to *medical record*)
- GetDocFromDos (get cover page from *medical record*)
- Edit (update number of drafts indicated by cover page)
- ReturnDoc (return cover page to *medical record*)
- ReturnDos (return *medical record* to *archive*)

25. It shall also be possible to request tables of contents (TOCs) and information about notes and medical records:
    (a) Request information about a *medical record*.
    (b) Request *person* TOC.
    (c) Request *person preference document*.
    (d) Request *medical record cart* TOC.
    (e) Request center TOC.
    (f) Request Get department TOC (based on center).
    (g) Request *medical record* ids based on search criteria.

13.3.5 Fitting

Fitting the system to already existing domains is beyond the scope of this Master Thesis. It shall be emphasized though, that several existing systems will have to be fitted, such as medical imaging databases and legacy systems in general (DK: Grønt System).

13.4 Interface Requirements

26. The paper-prototype displayed in appendix K page 199 shall serve as guidelines for the interface development.
27. The presentation of data in the client shall match the extracted *templates* as described in requirements item 15.
28. The presentation of a *medical record* shall match the color codes of categories in the domain and the graphical presentation shall imitate a paper medical record.
29. A recorder or dictaphone-like interface shall be available to the user in the client application.
30. The overall interface shall be tablet PC oriented, i.e. support the use of touch screens combined with pointing devices.
31. The interface shall support the execution of available *command macros* through buttons and context menus.
32. The interface shall support creation, editing and viewing of the different *notes*. This shall include the possibility of loading image files, such as `jpeg` and `bmp`.

13.5 Machine Requirements

33. Each system user shall be equipped with a *smart card*.
34. Windows Tablet PCs with wireless capabilities and smart card reader shall be available to the staff members.
35. Access points shall be set up covering the entire hospital with wireless system access.
36. Projectors shall be available in all rooms where medical images should be available.
37. A headset, possibly wireless, shall be available in order to record sound.

13.6 Glossary

Archive Refers to a specific index in the *directory*.

Category The *medical record* is divided into a number of categories each reserved for a special type of medical information.

Command macros A combination of the document system fundamental commands. The composition of the macros can be deducted by analyzing the required functionality (business processes).

Cover page A document contained in the outer dossier of a *medical record*. It holds all information present on the cover of a real-life *medical record*

Medical record A compilation of medical information about a *patient*. It is divided into *categories*. Each admitted *patient* has a medical record. The medical record is realized in the document system as a series of dossiers inside a master dossier.

Medical record cart Is represented by an index in the *directory*. The description of the index is similar to the department, i.e. the department index is equivalent to the cart.

Note Can be of either type admission note, ward round note, acute note, external notes or generic note.

Person An electronic representation of a user in the system.

Preference document is specific for each user and therefore also present on the their person. It holds user specific GUI initialization parameters.

Reference A reference is a copy of a given medical records cover page. From the reference it is possible to obtain an id of the *medical record* via commands in the document system.

Security certificate A certificate holds the necessary information in order to authenticate users and encrypt the data exchange.

Smart card A credit card sized memory bank that holds a *security certificate*.

Template A template is a definition of the transfer function $f(\mathcal{D}) \rightarrow \mathcal{C}$. It reveals how data $\mathcal{D}$ is formated to produce the presented information or contents $\mathcal{C}$

Chapter 14

Prototype Evaluation

Based on the requirement prescriptions a prototype of a 1st generation EMR system has been instantiated on top of the fundamental document system. This chapter will provide an overview of the finished prototype and illustrate how the fundamental document system is used to realize it. Additionally, it will briefly describe the basic steps required for migrating to a 2nd generation EMR system based on G-EPJ.

14.1 The First Generation EMR System

The EMR prototype has been implemented on top of the existing implementation of the document system. This work took us approximately eight weeks to complete. Four weeks were spent on the domain analysis, while two weeks were spent on the subsequent requirements specification. The last two weeks were spent on implementation based on the requirements. The implementation work is constituted by the following steps (the percentage indicates the approximate amount of time spent on the individual steps out of the total implementation time):

1. Creating a new server side business logic consisting of the macros of basic document commands as described during the systematic narrative of the EMR requirements. (15%)
2. Mapping the different structures of medical record notes into an XML Schema, and creating the necessary database tables and fields for storing these structures. (5%)
3. Creating a client side graphical user interface which presents the medical documents in a hospital domain oriented fashion. (80%)

During the GUI implementation step it has been attempted to imitate the layout of paper based medical records as closely as possible using the usability design principles of [21]. This involved colorized category tabs and formatted headlines and paragraphs in the medical notes as indicated by figures 14.1 and 14.2 page 106. Since one of the target client platforms is the Tablet PC much effort has been put into compensating for the potential lack of keyboard interfacing. This means that, apart from entering actual medical information, all operations on the medical records are carried out by clicking on various parts of the GUI. In addition, as prescribed by the requirements, dictaphone functionality has been integrated with the GUI so that recording and playback is performed by the client machine and stored in the EMR.

It is interesting to see that most of the implementation work has tilted towards GUI design. From an end-user's point of view this is a refreshing and ideal shift of focus as it allows for thorough usability studies of the part of the system that interacts with the user. In our opinion, it is a common mistake in software engineering that much less thought goes into GUI studies and development than on the server and database design. The graphical user interface is, after all, what the users will be confronted with so a system design should be centered around this and not vice versa.

Even though this is a non-optimized prototype with a fairly complex business logic, we have not detected any severe signs of performance degradation (that is, response times are in average less than one second). This strengthens our confidence in the document system design and the technologies used for implementing it. An extract of the source code of the EMR system business logic can be studied in appendix L page 203.

14.2 Migrating to Second Generation

The fundamental document system has been designed to allow continuous re-engineering of business process. In the EMR prototype, this re-engineering would be the future shift to 2nd generation electronic medical records.

This shift requires an analysis of the G-EPJ specification from a document oriented point of view, in many ways similar to the original domain analysis of the hospital. The individual document types of the G-EPJ specification, such as 'diagnostic notes', must be extracted. The new business processes, such as combining different document information by extracting 'focused information' for establishing a new diagnose document, must be listed and described, consult [34] for details.

Similar to the 1st generation EMR requirements development, the G-EPJ analysis can be used as a basis for formulating the new requirements of the 2nd generation EMR software. A combination of documents and dossiers describes the medical record structure and macros of basic document commands express the business processes.

By following this design pattern we expect it to be a relatively simple task to expand existing business logic and contents types. The only remaining task would be to design a new client application or extend the existing one to support the new business processes. The advantage is that both generations of medical records are kept in the same system allowing for backwards compatibility and possible reuse of information across EMR generations.

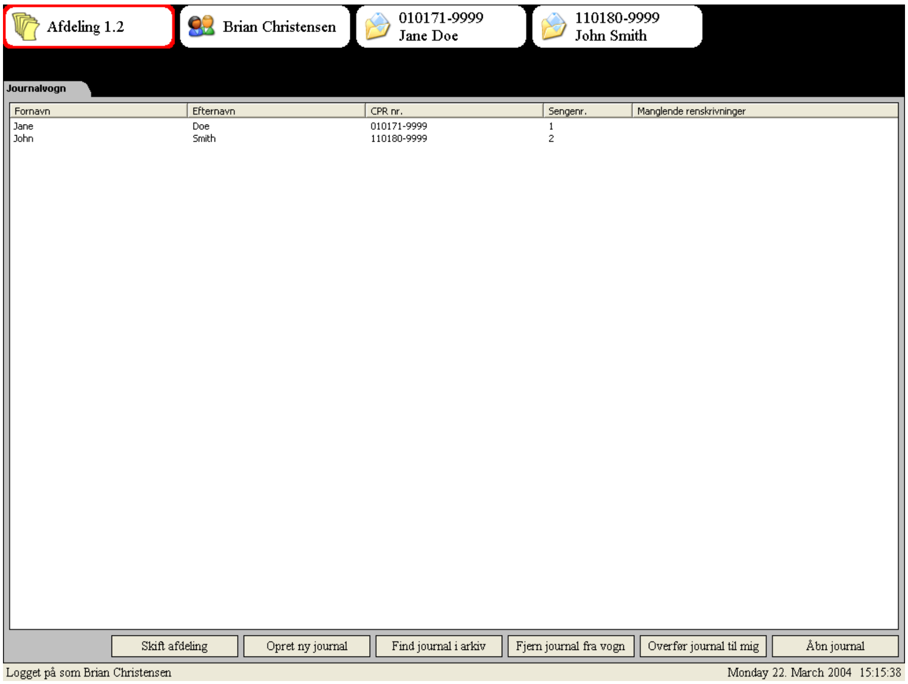


Figure 14.1: Department Medical Journal Cart

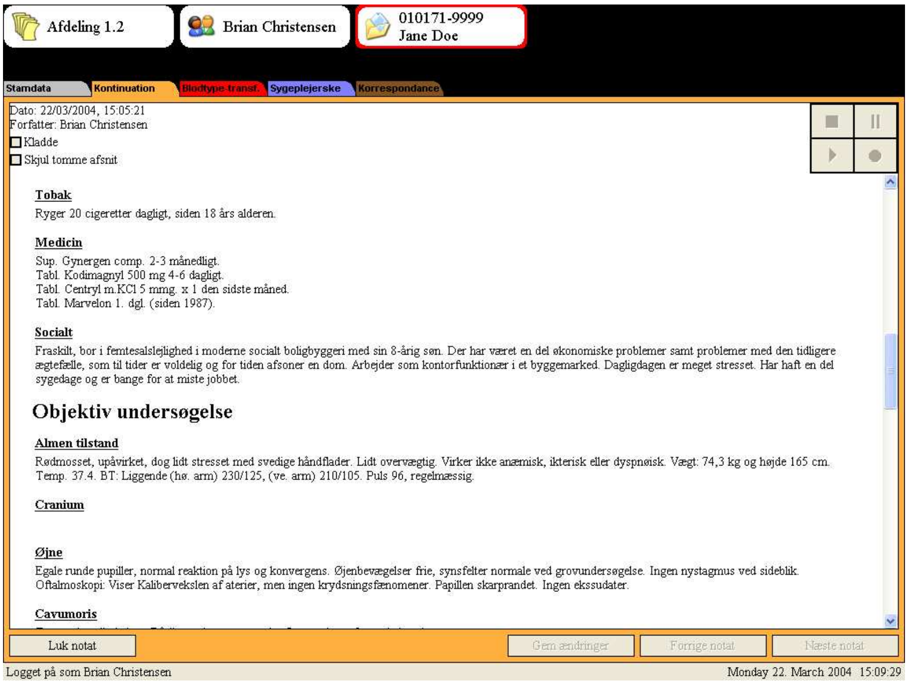


Figure 14.2: Admission Note in a Medical Record

Chapter 15

Conclusion

The purpose of the electronic medical record (EMR) part of the project has been to evaluate the strength of the basic document system when instantiating it for a specific domain. The aspects examined constitute:

1. How is the development process affected when using the terminology and principles of the basic paper document model?
2. Is it possible to digitize a specific paper based domain and adopt existing business processes using the basic document system as a foundation?
3. Is it feasible to initially adopt and implement a digitized version of the existing domain – and let business process re-engineering be a separate development step carried out some time after the transition to the digital domain?

The experiences gained in the process of tailoring the basic document system to the EMR domain have been through-out positive. The fact that most of the traditionally required features and technologies of distributed systems are embedded in the underlying platform have made it possible to focus exclusively on the hospital domain. This has resulted in a concise domain analysis and requirements specification focusing on central aspects of medical record management.

We have found that describing the hospital domain and EMR requirements was a relatively simple and intuitive process using the document-oriented terminology. The document model and terminology have provided the necessary building blocks for expressing the hospital and medical record structures, document types, and business processes in a systematic narrative closely resembling natural language – yet easily adoptable for requirements specification and implementation.

The design method shifts much of the implementation focus towards GUI development. This is a welcome change as it allows for thorough usability studies and considerations regarding how to use current technologies to support existing business processes, e.g. replacing the dictaphone with built-in recording facilities in the client hardware.

The alterations in business processes after digitizing the domain have been minimal. The obvious differences, such as several persons now being able to access the same record simultaneously, are considered beneficial extensions rather than changes – re-engineering – of the existing business processes. Based on this, we believe it has been shown that it is indeed possible to digitize a paper based document domain and adopt existing business processes. In theory, by building a new domain model and requirements specification as an extension of the document domain model. In practice, by creating a business logic and client application on top of the implementation of the document system. As a positive side note, it should be mentioned that no performance issues surfaced when instantiating the document system for the specific domain using this approach.

The time spent on the development of the EMR instantiation from the initial interviews to the finished implementation was approximately eight weeks. It is interesting to see that, out of these eight weeks, half of the time was spent on domain analysis while the remainder was divided between requirements specification and implementation. This illustrates the ease of digitizing a domain once it has been modelled in accordance to the document system. It demonstrates that it is not time consuming to initially digitize the domain without re-engineering – we believe this is also preferable as it indirectly provides the developers with a clear understanding of the existing domain before it is time to re-think business processes and make further use of the digitalization.

It has been considered how to perform later business process re-engineering once the domain is digitized, specifically by introducing 2nd generation medical record management in the prototype. In chapter 14 page 104 it is presented as a relatively simple new development phase consisting of further domain analysis, requirements specification and extensions to the existing implementation. The flexibility offered by the underlying document system provides the necessary means to make this a smooth transition that does not compromise backwards compatibility.

Part IV

SUMMARY

Chapter 16

Future Work

As previously indicated the work presented in this Master Thesis focuses on proof-of-concept. Taking the development to the next stage by creating a complete software product honoring all requirements of a professional EDMS solution requires further work – both technologically and scientifically.

16.1 Scientifically

It is difficult to assess the completeness of the paper document domain model as it has been derived through the authors' personal experiences when working with paper documents. Some aspects might have been left out and they would most likely surface when interviewing more people. This could help proving the model correct, or modifying or extending it if something is missing.

Another problem is that a lot of domains are already digitized to some degree and the question is whether the methodology provided by our EDMS framework is also applicable for developing EDMSs in these domains. As we see it, the main difference between an exclusively paper document domain compared to a partly digitized document domain is that some of the decisions when handling documents have been automated thereby moving them from the users to the computer. Using the document terminology this could be described as introducing computer controlled persons into the domain description. As the equivalences between the paper domain and the digital domain are made clear during the development of the domain model it should be possible to use it for describing both worlds – however, this is an exercise left for future studying.

Finally, it would be interesting to study document relations in the paper document domain. In the digital domain it is relatively easy, when using a database, to produce new documents from fragments of other documents. This could correspond to automating the actual world process of studying various documents and forming a general idea of:

1. how they relate to each other
2. the overall meaning they provide when combining them

It would be useful to explicitly model this process and include it in the terminology and business processes of the paper document domain model. Such an extension could further simplify the modelling of complex document relations which occur in the digital domain – for instance in the G-EPJ specification for future EMR systems.

16.2 Technologically

The future work which should be carried out technologically concerns optimization and testing. Now that a proof-of-concept has been conducted it is time to examine performance issues:

- The prototype needs extensive multiuser burn-in tests to verify the correctness of the design.
- The prototype needs optimization in several areas, such as database access and network communication.
- There are several minor implementation choices that would be preferable to change, such as streamlining how XML documents are handled in different parts of the implementation.
- Full system distribution needs to be implemented and tested to validate the design of the mirror concept.
- In the long run it is desirable to support web clients in addition to the existing solution.

Chapter 17

Our Experiences

Before we began working on the Master Thesis, it was our ambition to carry out a complete software development project. We wanted a special emphasis on domain analysis and requirements engineering using formal techniques but also on selecting current technologies for implementing and testing the design in practice. Naturally, this ambition has influenced our approach to the project, in particular the way we decided to document our findings.

We chose to adopt the triptych software engineering paradigm and the associated methodology presented in [6]. Following this principle, while conducting very constructive meetings and discussions with our supervisor regarding the method and areas to explore, helped focusing on the next natural step in the development process and resulted in very few iterations. In particular, our experiences gained from working with domain analysis has convinced us that this is an indispensable part of software engineering. We attribute most of our results to the initial work on domain analysis from which everything else was deducted. The complete methodology is an extensive procedure but all steps described in the methodology do not necessarily need to be followed meticulously at all times. It can be considered a guide for ensuring that all aspects are addressed either on paper, sketch, or in thought. The important thing is to know what to address.

In between working with the Master Thesis, we have spent time authoring an article and a business plan for the document oriented EDMS. The article (in Danish), see appendix M page 215, addresses the problems with regards to software development – with focus on document systems. Writing the article has helped us put the problems into perspective in a humanistic way ultimately giving us a better understanding. The business plan, see appendix N page 220, is a preliminary contribution to Venture Cup – a competition in generating and describing innovative ideas. Trying to author a business plan clearly demonstrated one of our weaker sides. Still it gave us an idea of what to expect if we want to continue with the idea presented in this report.

Chapter 18

Conclusion

The main thesis described during the introduction was:

> *Adopting the terminology and business processes of the paper document management domain results in an EDMS development platform which minimizes the language barrier between the domain specialists and the developers.*

We believe that the findings of parts II-III have shown this claim to be true. The language barrier problems between customer and developer presented in chapter 1 page 3 are all addressed by the EDMS development platform.

The development process of part II has resulted in an EDMS development platform, which provides the basic functionality expected from a document management system today. This includes versioning, structuring, distribution, authorization, and confidentiality. It has been combined with modern technologies providing database storage, secure transactions, and future-proof data encapsulation for exchanging information with foreign systems.

The development approach has resulted in a domain oriented terminology embedded in the EDMS development platform. It has also introduced a new document oriented analysis and design methodology associated to the terminology. This methodology extends existing development principles and it is to be used when tailoring the EDMS platform to a specific domain as demonstrated during the development of the EMR system.

These development principles and the associated paper document domain terminology can be used to describe a domain and express precise system requirements in a structured natural language. The structured requirements can be directly transformed to a business logic of the system thereby producing a domain specific EDMS in which the requirements are easy to identify and modify. The result is, in other words, a language that both developer and customer can use and relate to, and that the framework understands. The language focuses on the document domain while suppressing aspects foreign to the domain, but necessary when digitizing.

The EDMS platform was tested by designing and implementing a prototype of a domain oriented EMR system on top of the platform. We found the language provided by the platform to be fully adequate and intuitive to use when digitizing the EMR domain. As indicated in the findings of part III the new development principle keeps the focus entirely on the domain rather than on technologies and distributed system principles. Consequently, most of the time and energy could be spent on designing a graphical user interface inspired by the domain.

In the context of EMR systems we have evaluated the feasibility of initially adopting existing business processes and keep business process re-engineering as a separate development step. Although this subsequent re-engineering has not been carried out in practice on the EMR system, we have discussed how this would be done and considered the ramifications, such as issues with backwards compatibility and reuse of existing data. Based on these considerations and the relatively short amount of time it takes to digitize the domain using the EDMS platform, we have found it feasible to separate digitization and re-engineering when developing on top of the EDMS platform.

Naturally, it is too early to state that the document oriented development methodology is flawless. It was found suitable for digitizing the complex EMR domain and based on this we assume that it can be applied successfully when digitizing any paper document domain. However, as indicated in chapter 16 page 111, there are aspects which have been left for further studying. In particular, it would be interesting to see whether it is indeed possible to describe an already digitized domain using the paper document terminology.

In closing, we would like to comment on the chosen development method for domain analysis and requirements engineering. As mentioned in chapter 17 page 113, our experiences with using formal methods and domain analysis and requirements specification structuring have been overwhelmingly positive. The approach has provided a constant clear overview of the problem at hand and the next natural step in the development process. We are thoroughly convinced that this development strategy helps ensure correctness of the requirements while keeping the number of design and implementation iterations at a minimum.

Part V

APPENDIX

Appendix A

Original Problem

Der ønskes en realisering af et elektronisk patient journal system (EPJ-S) som bygger på Sundhedsstyrelsens rapport herom (EPJ). EPJ-S skal dels kunne håndtere de i EPJ beskrevne begreber, dels illustrere repræsentation og manipulation af ikke-traditionelle, dvs. ikke-tekstuelle dokumenter som f.eks. EKG (elektrokardiogrammer), MR (magnetisk resonans) skanninger, CT (computer tomografier), X-Rays, m.fl. Desuden skal EPJ-S kunne håndtere registrering af versioner af sådanne dokumenter: Originaler, kopier, samt redigeringer af originaler og kopier. Der skal lægges vægt på at EPJ-S designen relaterer sig til foreliggende oplæg vedr. domæne og kravspecifikationer, samt uddyber disse.

Appendix B

Encryption Principles

B.1 AsymmetricEncryption.rsl

```
scheme AsymmetricEncryption =
  class
    type
      Data,
      Key,
      Signature,
      KeyPair = Key × Key,
      HackKey = Key → Key,
      HackData = Data → Data

    value
      Encrypt : Data × Key → Data,
      Sign : Data × Key → Data × Signature,
      VerifySign : (Data × Signature) × Key → Bool

    axiom
      ∼(∃ f:HackKey •
        ∀ (publickey,privatekey):KeyPair •
          f(publickey) = privatekey ∨ f(privatekey) = f(publickey)),

      ∀ (publickey1,privatekey1):KeyPair,
          (publickey2,privatekey2):KeyPair •
       publickey1 = publickey2 ⇒ privatekey1 = privatekey2 ∧
       privatekey1 = privatekey2 ⇒ publickey1 = publickey2,

      ∼(∃ f:HackData •
        ∀ key:Key, data:Data • f(Encrypt(data,key)) = data),

      ∀ (publickey,privatekey):KeyPair, data:Data •
        Encrypt(Encrypt(data,publickey),privatekey) ≡ data,

      ∀ (publickey,privatekey):KeyPair, data:Data •
        VerifySign(Sign(data,privatekey),publickey)
  end
```

B.2 SymmetricEncryption.rsl

```
scheme SymmetricEncryption =
  class
    type
      Data,
      Key,
      HackData = Data → Data

    value
      Encrypt : Data × Key → Data,
      Decrypt : Data × Key → Data

    axiom
      ∀ secretkey:Key, data:Data •
        Decrypt(Encrypt(data,secretkey),secretkey) ≡ data,

      ∼(∃ f:HackData •
        ∀ secretkey:Key, data:Data •
          f(Encrypt(data,secretkey)) = data),
  end
```

Appendix C

DocSys – Draft Domain Specification

C.1 docsysoriginal.rsl

```
scheme DocSys3 =
  class
    type
      Place = Directory × Staff × Locations,
      PlaceId,
      Places = PlaceId →m Place,
      Person = DD-set,
      PersonId,
      Citizens = PersonId →m Person,
      Staff = PersonId →m Person,
      System = Places × Citizens × DocumentId-set × DossierId-set,
      Location = DD-set,
      LocationId,
      Locations = LocationId →m Location,
      Document,
      DocumentId,
      Documents = DocumentId →m Document,
      Dossier = Documents,
      Directory == mk_dir(DirName →m DD-set × Directory),
      DirName,
      DDId == mk_docId(DocumentId) | mk_dosId(DossierId),
      DD == mk_doc(doc:Document) | mk_dos(dos:Dossier),
      DossierId,
      Dossiers = DossierId →m Dossier,
      DocumentType == master|copy|version,
      Time,
      Info,
      Whereabouts == unknown | mk_citizen(cit:Person) |
      mk_staff(staff:Person, place:Place) | mk_location(loc:Location, place2:
     Place)

    type
      Cmd = CreaDoc | CreaDoss | Copy | Edit | Cit_Per | To_Doss |
            From_Doss | To_Dir | From_Dir | To_Loc | From_Loc |
            Shred | Per_Cit | Send | Return,
      CreaDoc :: per:Person plid:PlaceId lid:LocationId t:Time i:Info,
      CreaDoss :: per:Person plid:PlaceId lid:LocationId t:Time,
```

```
    Copy :: per:Person plid:PlaceId lid:LocationId t:Time doc:Document,
    Edit :: per:Person plid:PlaceId lid:LocationId t:Time doc:Document edit:
   FTE,
    FTE = (Info → Info) × (Info → Info)

  axiom
    ∀ (te,fe):FTE, i:Info • fe(te(i)) = i

  type
    Cit_Per :: cit:Person plid:PlaceId per:Person doc:Document,
    To_Doss :: person:Person plid:PlaceId doc:Document doss:Dossier,
    From_Doss :: person:Person plid:PlaceId doc:Document doss:Dossier,
    To_Dir :: person:Person plid:PlaceId dd:DD path:DirName*,
    From_Dir :: person:Person plid:PlaceId path:DirName* ddid:DDId,
    To_Loc :: person:Person plid:PlaceId lid:LocationId dd:DD,
    From_Loc :: person:Person plid:PlaceId lid:LocationId dd:DD,
    Shred :: person:Person plid:PlaceId dd:DD,
    Per_Cit :: person:Person plid:PlaceId doc:Document cit:Person,
    Send :: person:Person plid:PlaceId dd:DD person2:Person plid2:PlaceId,
    Return :: person:Person plid:PlaceId dd:DD person2:Person plid2:PlaceId

  value
    obs_Dir : PlaceId → Directory,
    obs_Staff : PlaceId → Staff,
    obs_Locations : PlaceId → Locations,
    obs_contents : Location → DD-set,
    obs_Id : Document → DocumentId,
    obs_Id : Dossier → DossierId,
    obs_Id : Person → PersonId,
    obs_Id : DD → DDId,
    obs_Type : Document → DocumentType,
    obs_Lenders : DD → Person*,

    obs_Ancestor : Document ~→ Document,
      pre obs_Type(doc) ≠ master

    obs_LocationHist : Document → LocationId,
    obs_PersonHist : Document → PersonId,
    obs_TimeHist : Document → Time,

    obs_DocumentHist : Document → Document*
    obs_DocumentHist(doc) ≡
      let doclist : Document* •
        ∀ n : Nat •
          n ≤ len doclist ∧ n > 1 ∧
          obs_Type(doclist(1)) = master ∧
          obs_Id(doc) = obs_Id(doclist(len doclist)) ∧
          obs_Type(doclist(n)) ≠ master  ⇒
            obs_Id(doclist(n-1)) = obs_Id(obs_Ancestor(doclist(n)))
      in
        doclist
      end,

    obs_BelongsToDir : DD → Bool,
    obs_DirInfo : DD ~→ DirName*,
      pre obs_BelongsToDir(dd)
    obs_AbsentFromDir : DD ~→ Bool,
      pre obs_BelongsToDir(dd)
    obs_Whereabouts : DD ~→ Whereabouts,
      pre obs_Absent(dd)
    obs_Information : Document → Info,
```

```

    RemoveDDFromDir : DDId → Directory,
    IsPathValid : DirName* × System → Bool,
    InsertDDIntoDir : DD × PlaceId × DirName* → Directory

  value
    M: Cmd → System → System
    M(cmd)(places,citizens,docids,dossids) ≡
      case cmd of
/*22*/   mk_CreaDoc(person,placeid,locationid,time,information) →
          let (dir, pers, locs) = places(placeid) in
            assert: person ∈ rng pers ∧
                    locationid ∈ locs
            let did:DocumentId • did ∉ docids in
              let doc:Document •
                obs_LocationHist(doc) = locationid ∧
                obs_TimeHist(doc) = time ∧
                obs_PersonHist(doc) = obs_Id(person) ∧
                obs_Information(doc) = information ∧
                obs_Id(doc) = did ∧
                ∼obs_BelongsToDir(mk_doc(doc)) ∧
                obs_Type(doc) = master in
                  (places † [placeid ↦ (dir,pers † [obs_Id(person) ↦ person ∪
                    {mk_doc(doc)}],locs)],citizens,docids ∪ {did},dossids)
              end
            end
          end,

/*23*/   mk_CreaDoss(person,placeid,locationid,time) →
          let (dir, pers, locs) = places(placeid) in
            assert: person ∈ rng pers ∧
                    locationid ∈ dom locs
            let did:DossierId • did ∉ dossids in
              let dossier:Dossier •
                dom dossier = {} ∧
                ∼obs_BelongsToDir(mk_dos(dossier)) ∧
                obs_Id(dossier) = did in
                  (places † [placeid ↦ (dir,pers † [obs_Id(person) ↦ person ∪
                    {mk_dos(dossier)}],locs)],citizens,docids, dossids ∪ {did
  })
              end
            end
          end,

/*24a*/  mk_Copy(person,placeid,locationid,time,document) →
          let (dir, pers, locs) = places(placeid) in
            assert: person ∈ rng pers ∧
                    locationid ∈ dom locs ∧
                    mk_doc(document) ∈ person
            let did:DocumentId • did ∉ docids in
              let doc:Document •
                obs_LocationHist(doc) = locationid ∧
                obs_TimeHist(doc) = time ∧
                obs_PersonHist(doc) = obs_Id(person) ∧
                obs_Information(doc) = obs_Information(document) ∧
                obs_Ancestor(doc) = document ∧
                obs_Type(doc) = copy ∧
                obs_Id(doc) = did ∧
                obs_DirInfo(mk_doc(doc)) = obs_DirInfo(mk_doc(document)) in
                  (places † [placeid ↦ (dir,pers † [obs_Id(person) ↦ person ∪
                    {mk_doc(doc)}],locs)],citizens,docids ∪ {did}, dossids)
              end
```

```
            end
          end,

/*24b*/   mk_Edit(person,placeid,locationid,time,document,(te,fe)) →
            let (dir, pers, locs) = places(placeid) in
              assert: person ∈ rng pers ∧
                        locationid ∈ locs ∧
                        obs_Type(document) ≠ master ∧
                        mk_doc(document) ∈ person
              let did:DocumentId • did ∉ docids in
                let doc:Document •
                  obs_LocationHist(doc) = locationid ∧
                  obs_TimeHist(doc) = time ∧
                  obs_PersonHist(doc) = obs_Id(person) ∧
                  obs_Ancestor(doc) = document ∧
                  obs_Type(doc) = version ∧
                  obs_Information(doc) = te(obs_Information(document)) ∧
                  obs_DirInfo(mk_doc(doc)) = obs_DirInfo(mk_doc(document)) in
                    (places † [placeid ↦ (dir,pers † [obs_Id(person) ↦ person ∪
                      {mk_doc(doc)}],locs)],citizens,docids ∪ {did}, dossids)
                end
              end
            end,

/*25*/    mk_Cit_Per(citizen,placeid,person,doc) →
            let (dir, pers, locs) = places(placeid) in
              assert: person ∈ rng pers ∧
                        citizen ∈ rng citizens ∧
                        mk_doc(doc) ∈ citizen ∧
                        obs_Type(doc) = master
              (places † [placeid ↦ (dir,pers †
                 [obs_Id(person) ↦ person ∪ {mk_doc(doc)}],locs)],
                 citizens † [obs_Id(citizen) ↦
                 citizen \ {mk_doc(doc)}],docids,dossids)
            end,

/*27a*/   mk_To_Doss(person,placeid,document,dossier) →
            let (dir, pers, locs) = places(placeid) in
              assert: person ∈ rng pers ∧
                        mk_doc(document) ∈ person ∧
                        mk_dos(dossier) ∈ person
              (places † [placeid ↦ (dir,pers † [obs_Id(person) ↦
                (person \ {mk_doc(document)}) ∪
                  {mk_dos(dossier † [obs_Id(document) ↦
                    document])}], locs)],
                      citizens, docids, dossids)
            end,

/*27b*/   mk_From_Doss(person,placeid,document,dossier) →
            let (dir, pers, locs) = places(placeid) in
              assert: person ∈ rng pers ∧
                        obs_Id(document) ∈ dom dossier ∧
                        mk_dos(dossier) ∈ person
              (places † [placeid ↦ (dir,pers † [obs_Id(person) ↦
                (person \ {mk_dos(dossier)}) ∪
                  {mk_doc(document),mk_dos(dossier \
                    {obs_Id(document)})}], locs)],
                      citizens,docids,dossids)
            end,

/*28*/   mk_To_Dir(person,placeid,dd,path) →
            let (dir, pers, locs) = places(placeid) in
```

```
          assert: person ∈ rng pers ∧
                    case dd of
                      mk_doc(doc) → obs_Id(doc) ∈ docids,
                      mk_dos(dos) → obs_Id(dos) ∈ dosids
                    end ∧
                    IsPathValid(path,(places,citizens,docids,dossids))

          (places † [placeid ↦
            (InsertDDIntoDir(dd,placeid,path),pers †
              [obs_Id(person) ↦ (person \ {dd})], locs)],
                citizens, docids, dossids)
        end,

/*29*/  mk_From_Dir(person,placeid,path,ddid) →
        let (dir, pers, locs) = places(placeid) in
          assert: person ∈ rng pers ∧
                    case ddid of
                      mk_docId(id) → obs(id) ∈ docids,
                      mk_dosId(id) → obs(id) ∈ dosids
                    end ∧
                    IsPathValid(path,(places,citizens,docids,dossids))
          let dd : DD • obs_Id(dd) = ddid in
          assert obs_DirInfo(dd) = path
            (places † [placeid ↦ (RemoveDDFromDir(ddid),pers †
              [obs_Id(person) ↦ (person ∪ {dd})], locs)],
                citizens, docids, dossids)
          end
        end,

/*30*/  mk_To_Loc(person,placeid,locationid,dd) →
        let (dir, pers, locs) = places(placeid) in
          assert: person ∈ rng pers ∧
                    locationid ∈ locs
                    dd ∈ person
          let locationcontents = locs(locationid) in
            (places † [placeid ↦ (dir,pers † [obs_Id(person) ↦ (person \ {
    dd})],
              locs † [locationid ↦ (locationcontents ∪
                {dd})])],citizens,
                  docids,dossids)
          end
        end,

/*31*/  mk_From_Loc(person,placeid,locationid,dd) →
        let (dir, pers, locs) = places(placeid) in
          assert: person ∈ rng pers ∧
                    locationid ∈ locs ∧
                    dd ∈ locs(locationid)
          let locationcontents = locs(locationid) in
            (places † [placeid ↦ (dir,pers †
              [obs_Id(person) ↦ (person ∪ {dd})],
                locs † [locationid ↦ (locationcontents \ {dd})])],
                  citizens, docids, dossids)
          end
        end,

/*32*/  mk_Send(person,placeid,dd,person2,placeid2) →
        let (dir, pers, locs) = places(placeid) in
          assert: person ∈ rng pers ∧
                    dd ∈ person
          let (dir2, pers2, locs2) = places(placeid2) in
            assert: person2 ∈ rng pers2 ∧
```

```
                (places † [placeid ↦ (dir,pers † [obs_Id(person) ↦
                  (person \ {dd})],locs), placeid2 ↦ (dir2, pers2 †
                    [obs_Id(person2) ↦ (person2 ∪ {dd})],locs)],
                      citizens,docids,dossids)
              end
              post hd obs_Lenders(dd) = person
            end,

/*34*/    mk_Return(person,placeid,dd,person2,placeid2) →
            let (dir, pers, locs) = places(placeid) in
              assert: person ∈ rng pers ∧
                        dd ∈ person
              let (dir2, pers2, locs2) = places(placeid2) in
                assert: person2 ∈ rng pers2 ∧
                          obs_Id(person2) = hd obs_Lenders(dd)
                (places † [placeid ↦ (dir,pers † [obs_Id(person) ↦
                  (person \ {dd})],locs), placeid2 ↦ (dir2, pers2 †
                    [obs_Id(person2) ↦ (person2 ∪ {dd})],locs)],
                      citizens,docids,dossids)
              end
            end,

/*35*/    mk_Shred(person, placeid, dd) →
            let (dir, pers, locs) = places(placeid) in
              assert: person ∈ rng pers ∧
                        dd ∈ person
              (places † [placeid ↦ (dir,pers †
                [obs_Id(person) ↦ (person \ {dd})], locs)],
                  citizens, docids, dossids)
            end,

/*36*/    mk_Per_Cit(person,placeid,document,citizen) →
            let (dir, pers, locs) = places(placeid) in
              assert: person ∈ rng pers ∧
                        mk_doc(document) ∈ person ∧
                        citizen ∈ rng citizens
              (places † [placeid ↦ (dir,pers † [obs_Id(person) ↦
                (person \ {mk_doc(document)})],locs)],
                  citizens † [obs_Id(citizen) ↦ (citizen ∪
                  {mk_doc(document)})],
                    docids, dossids)
            end
          end

end
```

Appendix D

DocSys – Domain Specification

D.1 docsystypes.rsl

```
scheme DocSysTypes =
  class
    type
      DocumentID,
      Document,
      DocumentType == master | copy | version,
      Ancestor == mk_did(did:DocumentID) | none,
      DossierID,
      DossierDescription,
      Dossier,

      PersonID,
      Person,
      Persons = PersonID →m Person,

      LocationID,
      Location,
      Locations = LocationID →m Location,

      PlaceID,

      Index,
      IndexDescription,
      DirContents,
      Directory == mk_dir(DirContents × (Index →m Directory)),
      DirPath == mk_dip(Index*) | none,

      Key,
      Keys = Key-set,
      Signature,
      Time,
      Contents,
      Envelope

    value
      obs_ID : Document → DocumentID,
      obs_Time : Document → Time,
```

```
      obs_PlaceID : Document → PlaceID,
      obs_Contents : Document → Contents,
      obs_Type : Document → DocumentType,
      obs_Creator : Document → PersonID,
      obs_DirMembership : Document → DirPath,
      obs_Signatures : Document → Signature-set,
      obs_Ancestor : Document → Ancestor,

      obs_Dossiers : Dossier → Dossier-set,
      obs_ID : Dossier → DossierID,
      obs_Documents : Dossier → Document-set,
      obs_Description : Dossier → DossierDescription,

      obs_ID : Person → PersonID,
      obs_Keys : Person → Keys,
      obs_Documents : Person → Document-set,
      obs_Dossiers : Person → Dossier-set,
      obs_Signature : Person → Signature,

      obs_Documents : Location → Document-set,
      obs_Dossiers : Location → Dossier-set,

      obs_Documents : DirContents → Document-set,
      obs_Dossiers : DirContents → Dossier-set,
      obs_Keys : DirContents → Keys,

      obs_Description : Index → IndexDescription
end
```

D.2 docsysbasics.rsl

```
DocSysTypes
scheme DocSysBasics =
  extend DocSysTypes with
  class
    value
      ∈ : Document × Dossier → Bool
      doc ∈ dos ≡
        doc ∈ obs_Documents(dos),

      ∈ : Document × Person → Bool
      doc ∈ pers ≡ (
        ∃! doslist:Dossier* •
          Xor(doc ∈ obs_Documents(pers),
              hd(doslist) ∈ obs_Dossiers(pers) ∧ doc ∈ recurseDossier(doslist))
     ),

      ∈ : Document × Location → Bool
      doc ∈ loc ≡ (
        ∃! doslist:Dossier* •
          Xor(doc ∈ obs_Documents(loc),
              hd(doslist) ∈ obs_Dossiers(loc) ∧
              doc ∈ recurseDossier(doslist))),

      ∈ : Document × DirContents → Bool
      doc ∈ dcontents ≡
        doc ∈ obs_Documents(dcontents),

      ∈ : Dossier × DirContents → Bool
      dos ∈ dcontents ≡
        dos ∈ obs_Dossiers(dcontents),
```

```

∪ : DirContents × Document-set → DirContents
dcontents ∪ ds ≡
  let ds1:DirContents •
    obs_Documents(ds1) = obs_Documents(dcontents) ∪ ds ∧
    obs_Dossiers(ds1) = obs_Dossiers(dcontents)
  in
    ds1
  end,

∪ : DirContents × Dossier-set → DirContents
dcontents ∪ ds ≡
  let ds1:DirContents •
    obs_Documents(ds1) = obs_Documents(dcontents) ∧
    obs_Dossiers(ds1) = obs_Dossiers(dcontents) ∪ ds
  in
    ds1
  end,

\ : DirContents × Document-set → DirContents
dcontents \ ds ≡
  let ds1:DirContents •
    obs_Documents(ds1) = obs_Documents(dcontents) \ ds ∧
    obs_Dossiers(ds1) = obs_Dossiers(dcontents)
  in
    ds1
  end,

\ : DirContents × Dossier-set → DirContents
dcontents \ ds ≡
  let ds1:DirContents •
    obs_Documents(ds1) = obs_Documents(dcontents) ∧
    obs_Dossiers(ds1) = obs_Dossiers(dcontents) \ ds
  in
    ds1
  end,

∪ : Person × Document-set → Person
per ∪ ds ≡
  let p:Person •
    obs_ID(p) = obs_ID(per) ∧
    obs_Signature(p) = obs_Signature(per) ∧
    obs_Keys(p) = obs_Keys(per) ∧
    obs_Documents(p) = obs_Documents(per) ∪ ds ∧
    obs_Dossiers(p) = obs_Dossiers(per)
  in
    p
  end,

∪ : Person × Dossier-set → Person
per ∪ ds ≡
  let p:Person •
    obs_ID(p) = obs_ID(per) ∧
    obs_Signature(p) = obs_Signature(per) ∧
    obs_Keys(p) = obs_Keys(per) ∧
    obs_Documents(p) = obs_Documents(per) ∧
    obs_Dossiers(p) = obs_Dossiers(per) ∪ ds
  in
    p
  end,

\ : Person × Document-set → Person
```

```
per \ ds ≡
  let p:Person •
    obs_ID(p) = obs_ID(per) ∧
    obs_Signature(p) = obs_Signature(per) ∧
    obs_Keys(p) = obs_Keys(per) ∧
    obs_Documents(p) = obs_Documents(per) \ ds ∧
    obs_Dossiers(p) = obs_Dossiers(per)
  in
    p
  end,

\ : Person × Dossier-set → Person
per \ ds ≡
  let p:Person •
    obs_ID(p) = obs_ID(per) ∧
    obs_Signature(p) = obs_Signature(per) ∧
    obs_Keys(p) = obs_Keys(per) ∧
    obs_Documents(p) = obs_Documents(per) ∧
    obs_Dossiers(p) = obs_Dossiers(per) \ ds
  in
    p
  end,

∪ : Location × Document-set → Location
loc ∪ ds ≡
  let l:Location •
    obs_Documents(l) = obs_Documents(loc) ∪ ds ∧
    obs_Dossiers(l) = obs_Dossiers(loc)
  in
    l
  end,

∪ : Location × Dossier-set → Location
loc ∪ ds ≡
  let l:Location •
    obs_Documents(l) = obs_Documents(loc) ∧
    obs_Dossiers(l) = obs_Dossiers(loc) ∪ ds
  in
    l
  end,

\ : Location × Document-set → Location
loc \ ds ≡
  let l:Location •
    obs_Documents(l) = obs_Documents(loc) \ ds ∧
    obs_Dossiers(l) = obs_Dossiers(loc)
  in
    l
  end,

\ : Location × Dossier-set → Location
loc \ ds ≡
  let l:Location •
    obs_Documents(l) = obs_Documents(loc) ∧
    obs_Dossiers(l) = obs_Dossiers(loc) \ ds
  in
    l
  end,

∈ : Document × Directory → Bool
doc ∈ dir ≡ (
  ∃! dirpath:Index*, doslist:Dossier* •
```

```
      Xor(doc ∈ recurseDir(dir,dirpath),
          hd(doslist) ∈ recurseDir(dir,dirpath) ∧
          doc ∈ recurseDossier(doslist))),

  ∈ : Dossier × Dossier → Bool
  dos1 ∈ dos ≡
    dos1 ∈ obs_Dossiers(dos),

  ∈ : Dossier × Person → Bool
  dos ∈ pers ≡ (
    ∃! doslist:Dossier* •
      hd(doslist) ∈ pers ∧ obs_ID(dos) = obs_ID(recurseDossier(doslist))),

  ∈ : Dossier × Location → Bool
  dos ∈ loc ≡ (
    ∃! doslist:Dossier* •
       hd(doslist) ∈ loc ∧ obs_ID(dos) = obs_ID(recurseDossier(doslist))),

  ∈ : Dossier × Directory → Bool
  dos ∈ dir ≡ (
    ∃! dirpath:Index*, doslist:Dossier* •
      hd(doslist) ∈ recurseDir(dir,dirpath) ∧
        obs_ID(dos) = obs_ID(recurseDossier(doslist)))

value
  recurseDir : Directory × Index* → DirContents
  recurseDir(dir, dirpath) ≡
    let mk_dir(dcontents, dirs) = dir in
      if len dirpath = 0 then dcontents
      else
        recurseDir(dirs(hd(dirpath)),tl(dirpath))
      end
    end
    pre let mk_dir(dcontents, dirs) = dir in
      len dirpath = 0 ∨ hd(dirpath) ∈ dom dirs
    end,

  recurseDossier : Dossier* → Dossier
  recurseDossier(doslist) ≡
    if len doslist = 1 then hd(doslist)
    else
      recurseDossier(tl(doslist))
    end
    pre len doslist = 1 ∨ hd(tl(doslist)) ∈ hd(doslist),

  updateDir : Directory × Index* × DirContents → Directory
  updateDir(dir, dirpath, dcontents) ≡
    let mk_dir(dcontents1, dirs) = dir in
      if len dirpath = 0 then mk_dir(dcontents, dirs)
      else
        mk_dir(dcontents, dirs † [hd(dirpath) ↦
          updateDir(dirs(hd(dirpath)), tl(dirpath), dcontents)])
      end
    end
    pre let mk_dir(dcontents1, dirs) = dir in
      len dirpath = 0 ∨ hd(dirpath) ∈ dom dirs
    end,

  Xor : Bool × Bool → Bool
  Xor(arg1,arg2) ≡(
    arg1 ≠ arg2),
```

```
      indexExists : Directory × Index* → Bool
      indexExists(dir, dirpath) ≡
        let mk_dir(dcontents, dirs) = dir in
          if len dirpath = 0 then
            true
          elsif hd(dirpath) ∉ dom dirs then
            false
          else
              indexExists(dirs(hd(dirpath)),tl(dirpath))
          end
        end,

      assert : Bool → Unit
      assert(criteria) ≡
        if ∼criteria then
          chaos
        end
  end
```

D.3 pdocsystypes.rsl

```
docsysbasics
scheme pDocSysTypes =
  extend DocSysBasics with
  class
    type
      System′ = Places × DocumentID-set × DossierID-set,
      System = {| w:System′ • wf_system(w) |}

    value
      wf_system : System′ → Bool

    type
      PlaceMembership == mk_plm(PlaceID) | none,
      Place = Directory × Persons × Locations × Keys,
      Places = PlaceID →m Place,

      FTE = (Contents → Contents) × (Contents → Contents)

    value
      obs_PlaceMembership : Document → PlaceMembership

    value
      PersonBorn: System × Place × Person × PersonID → System,
      PersonDeceased: System × Person → System,
      IssuePlacePermit: System × PlaceID → System,
      SuspensePlacePermit: System × PlaceID → System,

      MakeKey: System × Place → System,
      DestroyKey: System × Place × Key → System,
      CopyKey: System × Place × Person × Key → System,
      RemoveKey: System × Place × Person × Key → System,
      CreateDirIndex: System × Place × Index* → System,
      DeleteDirIndex: System × Place × Index* → System,
      BuildLocation: System × Place × Location → System,
      DestroyLocation: System × Place × Location → System
  end
```

D.4 pdocsysbasics.rsl

```
pdocsystypes
scheme pDocSysBasics =
  extend pDocSysTypes with
  class
    value
      ∪ : Dossier × Document-set → Dossier
      dos ∪ ds ≡
        let d:Dossier •
          obs_ID(d) = obs_ID(dos) ∧
          obs_Description(d) = obs_Description(dos) ∧
          obs_Documents(d) = obs_Documents(dos) ∪ ds ∧
          obs_Dossiers(d) = obs_Dossiers(dos)
        in d end,

      \ : Dossier × Document-set → Dossier
      dos \ ds ≡
        let d:Dossier •
          obs_ID(d) = obs_ID(dos) ∧
          obs_Description(d) = obs_Description(dos) ∧
          obs_Documents(d) = obs_Documents(dos) \ ds ∧
          obs_Dossiers(d) = obs_Dossiers(dos)
        in d end,

      ∪ : Dossier × Dossier-set → Dossier
      dos ∪ doss ≡
        let d:Dossier •
          obs_ID(d) = obs_ID(dos) ∧
          obs_Description(d) = obs_Description(dos) ∧
          obs_Documents(d) = obs_Documents(dos) ∧
          obs_Dossiers(d) = obs_Dossiers(dos) ∪ doss
        in d end,

      \ : Dossier × Dossier-set → Dossier
      dos \ doss ≡
        let d:Dossier •
          obs_ID(d) = obs_ID(dos) ∧
          obs_Description(d) = obs_Description(dos) ∧
          obs_Documents(d) = obs_Documents(dos) ∧
          obs_Dossiers(d) = obs_Dossiers(dos) \ doss
        in d end,

      addSignature : Document × Person → Document
      addSignature(doc, person) ≡
        let d:Document •
          obs_ID(d) = obs_ID(doc) ∧
          obs_Time(d) = obs_Time(doc) ∧
          obs_PlaceID(d) = obs_PlaceID(doc) ∧
          obs_Contents(d) = obs_Contents(doc) ∧
          obs_Type(d) = obs_Type(doc) ∧
          obs_Creator(d) = obs_Creator(doc) ∧
          obs_Signatures(d) = obs_Signatures(doc) ∪ {obs_Signature(person)} ∧
          obs_PlaceMembership(d) = obs_PlaceMembership(doc) ∧
          obs_DirMembership(d) = obs_DirMembership(doc) ∧
          obs_Ancestor(d) = obs_Ancestor(doc)
        in d end,

      setMembership : Document × PlaceMembership × DirPath → Document
      setMembership(doc, plm, dirpath) ≡
        if (dirpath = none ∧ plm = none) ∨ (obs_DirMembership(doc) = none ∧
          obs_PlaceMembership(doc) = none) then
          let d:Document •
            obs_ID(d) = obs_ID(doc) ∧
```

```
          obs_Time(d) = obs_Time(doc) ∧
          obs_PlaceID(d) = obs_PlaceID(doc) ∧
          obs_Contents(d) = obs_Contents(doc) ∧
          obs_Type(d) = obs_Type(doc) ∧
          obs_Creator(d) = obs_Creator(doc) ∧
          obs_Signatures(d) = obs_Signatures(doc) ∧
          obs_DirMembership(d) = dirpath ∧
          obs_PlaceMembership(d) = plm ∧
          obs_Ancestor(d) = obs_Ancestor(doc)
        in d end
      else doc end,

    setMembership : Dossier × PlaceMembership × DirPath → Dossier
    setMembership(dos, plm, dirpath) as d
      post (
        (all doc:Document • doc ∈ d ⇒
          doc = setMembership(doc, plm, dirpath)) ∧
        (all dos:Dossier • dos ∈ d ⇒
          dos = setMembership(dos, plm, dirpath))
      )
  end
```

D.5 pdocsyswf.rsl

```
pdocsysbasics
scheme pDocSysWF =
  extend pDocSysBasics with
  class
    value
      wf_doc : System′ → Bool
      wf_doc((places,docids_in_use,dosids_in_use)) ≡ (
        ∀ doc,doc2:Document •
          obs_ID(doc) ∈ docids_in_use ⇒ (
          obs_ID(doc) = obs_ID(doc2) ⇒ doc = doc2 ∧
          (∃! (dir,pers,locs,keys):Place, person:Person, loc:Location •
            (dir,pers,locs,keys) ∈ rng places ∧
            (Xor(Xor(person ∈ rng pers ∧ doc ∈ person,
                     loc ∈ rng locs ∧ doc ∈ loc),
                     doc ∈ dir)))))
    value
      wf_dos : System′ → Bool
      wf_dos((places,docids_in_use,dosids_in_use)) ≡ (
        ∀ dos,dos2:Dossier •
          obs_ID(dos) ∈ dosids_in_use ∧
          (obs_ID(dos) = obs_ID(dos2) ⇒ dos = dos2) ∧
          (∃! (dir,pers,locs,keys):Place, person:Person, loc:Location •
            (dir,pers,locs,keys) ∈ rng places ∧
            (Xor(Xor(person ∈ rng pers ∧ dos ∈ person,
                         loc ∈ rng locs ∧ dos ∈ loc),
                         dos ∈ dir))))
    value
      wf_pers : System′ → Bool
      wf_pers((places,docids_in_use,dosids_in_use)) ≡ (
        ∀ pers,pers2:Person •
          (obs_ID(pers) = obs_ID(pers2) ⇒ pers = pers2) ∧
          (obs_Signature(pers) = obs_Signature(pers2) ⇒ pers = pers2) ∧
          (∃! (dir,pers1,locs,keys):Place •
            (dir,pers1,locs,keys) ∈ rng places ∧ pers ∈ rng pers1))

    axiom
      ∀ w:System′ •
```

```
        wf_system(w) ≡ (wf_doc(w) ∧ wf_dos(w) ∧ wf_pers(w))
end
```

D.6 pdocsyscmds.rsl

```
pdocsyswf
scheme pDocSysCmds =
  extend pDocSysWF with
  class
    type
      Command = CreateDoc
              | CreateDos
              | Copy
              | Edit
              | Shred
              | DisposeOfDos
              | GetDocFromDos
              | PutDocInDos
              | GetDosFromDos
              | PutDosInDos
              | GetDocFromDir
              | PutDocInDir
              | GetDosFromDir
              | PutDosInDir
              | GetDocFromLoc
              | GetDosFromLoc
              | PutDocInLoc
              | PutDosInLoc
              | SignDocument
              | ResetMembership
              | SendDoc
              | SendDos

    type
      CreateDoc ::
        ref_person : Person
        ref_PlaceID : PlaceID
        ref_time : Time
        ref_contents : Contents,

      CreateDos ::
        ref_person : Person
        ref_PlaceID : PlaceID,

      Copy ::
        ref_person : Person
        ref_PlaceID : PlaceID
        ref_time : Time
        ref_doc : Document,

      Edit ::
        ref_person : Person
        ref_PlaceID : PlaceID
        ref_time : Time
        ref_doc : Document
        ref_edition : FTE,

      Shred ::
        ref_person : Person
        ref_PlaceID : PlaceID
        ref_doc : Document,
```

```

    DisposeOfDos ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_dos : Dossier,

    GetDocFromDos ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_dos : Dossier
      ref_doc : Document,

    PutDocInDos ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_dos : Dossier
      ref_doc : Document,

    GetDosFromDos ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_dos : Dossier
      ref_doc : Dossier,

    PutDosInDos ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_dos : Dossier
      ref_doc : Dossier,

    GetDocFromDir ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_dirpath : Index*
      ref_docid : DocumentID,

    PutDocInDir ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_dirpath : Index*
      ref_doc : Document,

    GetDosFromDir ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_dirpath : Index*
      ref_dosid : DossierID,

    PutDosInDir ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_dirpath : Index*
      ref_dos : Dossier,

    GetDocFromLoc ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_locid : LocationID
      ref_docid : DocumentID,

    GetDosFromLoc ::
      ref_person : Person
```

```
      ref_PlaceID : PlaceID
      ref_locid : LocationID
      ref_dosid : DossierID,

    PutDocInLoc ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_locid : LocationID
      ref_doc : Document,

    PutDosInLoc ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_locid : LocationID
      ref_dos : Dossier,

    SignDocument ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_doc : Document,

    SendDoc ::
      ref_sender : Person
      ref_origin : PlaceID
      ref_env : Envelope
      ref_reciever : PersonID
      ref_dest : PlaceID
      ref_doc : Document,

    SendDos ::
      ref_sender : Person
      ref_origin : PlaceID
      ref_env : Envelope
      ref_reciever : PersonID
      ref_dest : PlaceID
      ref_doc : Dossier,

    ResetMembership ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_doc : Document

  value
    M: Command → System → System
    M(cmd)(places, docids, dosids) ≡
      case cmd of
        mk_CreateDoc(person, plid, time, contents) →
          let (dir,pers,locs,keys) = places(plid) in
            assert(person ∈ rng pers);
            let did:DocumentID • did ∉ docids in
              let doc:Document •
                obs_ID(doc) = did ∧
                obs_Time(doc) = time ∧
                obs_Contents(doc) = contents ∧
                obs_Type(doc) = master ∧
                obs_Creator(doc) = obs_ID(person) ∧
                obs_PlaceID(doc) = plid ∧
                obs_Signatures(doc) = {} ∧
                obs_DirMembership(doc) = none ∧
                obs_PlaceMembership(doc) = none ∧
                obs_Ancestor(doc) = none
              in
```

```
        (places † [plid ↦
          (dir, pers † [obs_ID(person) ↦
            person ∪ {doc}],
          locs, keys)], docids ∪ {did}, dosids)
      end
    end
  end,

mk_CreateDos(person, plid) →
  let (dir,pers,locs,keys) = places(plid) in
    assert(person ∈ rng pers);
    let dosid:DossierID • dosid ∉ dosids in
      let dos:Dossier •
        obs_ID(dos) = dosid ∧
        obs_Documents(dos) = {} ∧
        obs_Dossiers(dos) = {}
      in
        (places † [plid ↦
          (dir, pers † [obs_ID(person) ↦
            person ∪ {dos}],
          locs, keys)], docids, dosids ∪ {dosid})
      end
    end
  end,

mk_Copy(person, plid, time, doc) →
  let (dir,pers,locs,keys) = places(plid) in
    assert(person ∈ rng pers ∧
           doc ∈ obs_Documents(person));
    let did:DocumentID • did ∉ docids in
      let cpy:Document •
        obs_ID(cpy) = did ∧
        obs_Time(cpy) = time ∧
        obs_Contents(cpy) = obs_Contents(doc) ∧
        obs_Type(cpy) = copy ∧
        obs_Creator(cpy) = obs_ID(person) ∧
        obs_PlaceID(cpy) = plid ∧
        obs_Signatures(cpy) = obs_Signatures(doc) ∧
        obs_DirMembership(cpy) = none ∧
        obs_PlaceMembership(cpy) = none ∧
        if obs_Type(doc) = copy then
          obs_Ancestor(cpy) = obs_Ancestor(doc)
        else
          obs_Ancestor(cpy) = mk_did(obs_ID(doc))
        end
      in
        (places † [plid ↦
          (dir, pers † [obs_ID(person) ↦
            person ∪ {cpy}],
          locs, keys)], docids ∪ {did}, dosids)
      end
    end
  end,

mk_Edit(person, plid, time, document, (te,fe)) →
  let (dir,pers,locs,keys) = places(plid) in
    assert(person ∈ rng pers ∧
           document ∈ obs_Documents(person));
    let doc:Document •
      obs_ID(doc) = obs_ID(document) ∧
      obs_Time(doc) = time ∧
      obs_Contents(doc) = te(obs_Contents(document)) ∧
```

```
      obs_Type(doc) = version ∧
      obs_Creator(doc) = obs_ID(person) ∧
      obs_PlaceID(doc) = plid ∧
      obs_Signatures(doc) = {} ∧
      obs_DirMembership(doc) = obs_DirMembership(document) ∧
      obs_PlaceMembership(doc) = obs_PlaceMembership(document) ∧
      obs_Ancestor(doc) = obs_Ancestor(document)
    in
      (places † [plid ↦
        (dir, pers † [obs_ID(person) ↦
          (person \ {document}) ∪ {doc}],
        locs, keys)], docids, dosids)
    end
  end,

mk_DisposeOfDos(person, plid, dos) →
  let (dir,pers,locs,keys) = places(plid) in
    assert(person ∈ rng pers ∧
           dos ∈ obs_Dossiers(person));
    (places † [plid ↦
      (dir, pers † [obs_ID(person) ↦
        person \ {dos}],
      locs, keys)], docids, dosids \ {obs_ID(dos)})
  end,

mk_Shred(person, plid, doc) →
  let (dir,pers,locs,keys) = places(plid) in
    assert(person ∈ rng pers ∧
           doc ∈ obs_Documents(person));
    (places † [plid ↦
      (dir, pers † [obs_ID(person) ↦
        person \ {doc}],
      locs, keys)], docids \ {obs_ID(doc)}, dosids)
  end,

mk_GetDocFromDos(person, plid, dos, doc) →
  let (dir,pers,locs,keys) = places(plid) in
    assert(person ∈ rng pers ∧
           dos ∈ obs_Dossiers(person) ∧
           doc ∈ obs_Documents(dos));
    (places † [plid ↦
      (dir, pers † [obs_ID(person) ↦
        (person \ {dos}) ∪
          {dos \ {doc}} ∪
          {doc}], locs, keys)],
      docids, dosids)
  end,

mk_PutDocInDos(person, plid, dos, doc) →
  let (dir,pers,locs,keys) = places(plid) in
    assert(person ∈ rng pers ∧
           dos ∈ obs_Dossiers(person) ∧
           doc ∈ obs_Documents(person));
    (places † [plid ↦
      (dir, pers † [obs_ID(person) ↦
        (person \ {dos}) \ {doc} ∪
          {dos ∪ {doc}}], locs, keys)],
      docids, dosids)
  end,

mk_GetDosFromDos(person, plid, dos, dos1) →
  let (dir,pers,locs,keys) = places(plid) in
```

```
          assert(person ∈ rng pers ∧
                 dos ∈ obs_Dossiers(person) ∧
                 dos1 ∈ obs_Dossiers(dos));
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              (person \ {dos}) ∪
                {dos \ {dos1}} ∪
                {dos1}], locs, keys)],
            docids, dosids)
        end,

      mk_PutDosInDos(person, plid, dos, dos1) →
        let (dir,pers,locs,keys) = places(plid) in
          assert(person ∈ rng pers ∧
                 dos ∈ obs_Dossiers(person) ∧
                 dos1 ∈ obs_Dossiers(person));
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              (person \ {dos}) \ {dos1} ∪
                {dos ∪ {dos1}}], locs, keys)],
            docids, dosids)
        end,

      mk_GetDocFromDir(person, plid, dirpath, docid) →
        let (dir,pers,locs,keys) = places(plid) in
          let doc:Document • obs_ID(doc) = docid in
            assert(person ∈ rng pers ∧
                   indexExists(dir,dirpath) ∧
                   doc ∈ recurseDir(dir,dirpath) ∧
                   obs_Keys(recurseDir(dir,dirpath)) ⊂ obs_Keys(person));
              (places † [plid ↦
                (updateDir(dir,dirpath,recurseDir(dir,dirpath) \
                   {doc}),
                 pers † [obs_ID(person) ↦
                   (person ∪ {doc})],locs, keys)],
                 docids, dosids)
          end
        end,

      mk_PutDocInDir(person, plid, dirpath, doc) →
        let (dir,pers,locs,keys) = places(plid) in
          assert(person ∈ rng pers ∧
                 doc ∈ obs_Documents(person) ∧
                 indexExists(dir,dirpath) ∧
                 obs_Keys(recurseDir(dir,dirpath)) ⊂ obs_Keys(person));
            let destination = recurseDir(dir,dirpath) in
              (places † [plid ↦
                (updateDir(dir,dirpath,destination ∪
                   {setMembership(doc,
                      mk_plm(plid),mk_dip(dirpath))}),
                 pers † [obs_ID(person) ↦
                   (person \ {doc})],locs, keys)],
                 docids, dosids)
             end
        end,

      mk_GetDosFromDir(person, plid, dirpath, dosid) →
        let (dir,pers,locs,keys) = places(plid) in
          let dos:Dossier • obs_ID(dos) = dosid in
            assert(person ∈ rng pers ∧
                   indexExists(dir,dirpath) ∧
                   dos ∈ recurseDir(dir,dirpath) ∧
```

```
                obs_Keys(recurseDir(dir,dirpath)) ⊂ obs_Keys(person));
            (places † [plid ↦
              (updateDir(dir,dirpath,recurseDir(dir,dirpath) \
                {dos}),
               pers † [obs_ID(person) ↦
                 (person ∪ {dos})],locs, keys)],
               docids, dosids)
        end
      end,

    mk_PutDosInDir(person, plid, dirpath, dos) →
      let (dir,pers,locs,keys) = places(plid) in
        assert(person ∈ rng pers ∧
               dos ∈ obs_Dossiers(person) ∧
               indexExists(dir,dirpath) ∧
               obs_Keys(recurseDir(dir,dirpath)) ⊂ obs_Keys(person));
          let destination = recurseDir(dir, dirpath) in
            (places † [plid ↦
              (updateDir(dir,dirpath,destination ∪
                {setMembership(dos,
                   mk_plm(plid),mk_dip(dirpath))}),
               pers † [obs_ID(person) ↦
                 (person \ {dos})],locs, keys)],
               docids, dosids)
          end
      end,

    mk_GetDocFromLoc(person, plid, locid, docid) →
      let (dir,pers,locs,keys) = places(plid) in
        assert(person ∈ rng pers ∧
               locid ∈ dom locs ∧
               (∃ doc:Document •
                 obs_ID(doc) = docid ∧
                 doc ∈ obs_Documents(locs(locid))));
        let doc:Document • obs_ID(doc) = docid,
            loccont = locs(locid)
          in
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              (person ∪ {doc})],
            (locs † [locid ↦
              (loccont \ {doc})]), keys)],
            docids, dosids)
        end
      end,

    mk_PutDocInLoc(person, plid, locid, doc) →
      let (dir,pers,locs,keys) = places(plid) in
        assert(person ∈ rng pers ∧
               locid ∈ locs ∧
               doc ∈ obs_Documents(person));
        let loccontents = locs(locid) in
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              (person \ {doc})],
            (locs † [locid ↦
              (loccontents ∪ {doc})]), keys)],
            docids, dosids)
        end
      end,

    mk_GetDosFromLoc(person, plid, locid, dosid) →
```

```
      let (dir,pers,locs,keys) = places(plid) in
        assert(person ∈ rng pers ∧
               locid ∈ dom locs ∧
               (∃ dos:Dossier •
                 obs_ID(dos) = dosid ∧
                 dos ∈ obs_Dossiers(locs(locid))));
        let dos:Dossier • obs_ID(dos) = dosid,
            loccont = locs(locid)
          in
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              (person ∪ {dos})],
            (locs † [locid ↦
              (loccont \ {dos})]), keys)],
            docids, dosids)
        end
      end,

    mk_PutDosInLoc(person, plid, locid, dos) →
      let (dir,pers,locs,keys) = places(plid) in
        assert(person ∈ rng pers ∧
               locid ∈ dom locs ∧
               dos ∈ obs_Dossiers(person));
        let loccontents = locs(locid) in
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              (person \ {dos})],
            (locs † [locid ↦
              (loccontents ∪ {dos})]), keys)],
            docids, dosids)
        end
      end,

    mk_SignDocument(person, plid, doc) →
      let (dir,pers,locs,keys) = places(plid) in
        assert(person ∈ rng pers ∧
                doc ∈ obs_Documents(person));
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              ((person \ {doc}) ∪
                {addSignature(doc,person)})],
             locs, keys)],
            docids, dosids)
      end,

    mk_SendDoc(person, plid_org, env, pid_dst, plid_dst, doc) →
      let (dir,pers,locs,keys) = places(plid_org),
        (dir_dst,pers_dst,locs_dst,keys_dst) = places(plid_dst),
        person_dst : Person • obs_ID(person_dst) = pid_dst
      in
        assert(person ∈ rng pers ∧
               pid_dst ∈ dom pers_dst ∧
               doc ∈ obs_Documents(person));
        (places † [plid_org ↦
          (dir, pers † [obs_ID(person) ↦
            (person \ {doc})],
          locs, keys),
         plid_dst ↦
          (dir_dst, pers_dst † [ pid_dst ↦
            (person_dst ∪ {doc})],
           locs_dst, keys_dst)],
        docids, dosids)
```

```
          end,

        mk_SendDos(person, plid_org, env, pid_dst, plid_dst, dos) →
          let (dir,pers,locs,keys) = places(plid_org),
            (dir_dst,pers_dst,locs_dst,keys_dst) = places(plid_dst),
            person_dst : Person • obs_ID(person_dst) = pid_dst
          in
            assert(person ∈ rng pers ∧
                   pid_dst ∈ dom pers_dst ∧
                   dos ∈ obs_Dossiers(person));
            (places † [plid_org ↦
              (dir, pers † [obs_ID(person) ↦
                (person \ {dos})],
              locs, keys),
             plid_dst ↦
              (dir_dst, pers_dst † [ pid_dst ↦
                (person_dst ∪ {dos})],
               locs_dst, keys_dst)],
            docids, dosids)
          end,

        mk_ResetMembership(person, plid, doc) →
          let (dir,pers,locs,keys) = places(plid) in
            assert(person ∈ rng pers ∧
                   doc ∈ obs_Documents(person));
            (places † [plid ↦
              (dir, pers † [obs_ID(person) ↦
                (person \ {doc}) ∪
                  {setMembership(doc,none,none)}],
              locs, keys)],
            docids, dosids)
          end
        end
end
```

Appendix E

DocSys – Requirements Specification

E.1 edocsystypes.rsl

```
docsysbasics
scheme eDocSysTypes =
  extend DocSysBasics with
  class
    type
      System′ = Places × DocumentID-set × DossierID-set × ExportID-set,
      System = {| w:System′ • wf_system(w) |}

    value
      wf_system : System′ → Bool

    type
      ExportID,
      RecycleBin,
      Place = Directory × Persons × Locations × RecycleBin × Keys,
      Places = PlaceID →m Place,

      FTE = (Contents → Contents) × (Contents → Contents),
      PersonDossier = Person × Dossier*,
      DossierMembership == mk_did(DossierID) | none

    value
      obs_Events : Document → Event*,
      obs_DossierMembership : Document → DossierMembership,
      obs_CommandLocks : Document → CommandLocks,

      obs_PlaceID : Dossier → PlaceID,
      obs_DirMembership : Dossier → DirPath,
      obs_DossierMembership : Dossier → DossierMembership,
      obs_CommandLocks : Dossier → CommandLocks,
      obs_Documents : RecycleBin → Document-set,
      obs_Dossiers : RecycleBin → Dossier-set

    value
      obs_Group : Document × DocumentID-set → Document-set
      obs_Group(doc,docids) as c
        post
```

```
        doc ∈ c ∧
        (all doc1,doc2:Document •
          obs_ID(doc1) ∈ docids ∧ obs_ID(doc2) ∈ docids ⇒
            if doc1 ∈ c ∧
               (obs_Type(doc2) = version ∧
               mk_did(obs_ID(doc1)) = obs_Ancestor(doc2))
               ∨
               (obs_Type(doc1) = version ∧
               mk_did(obs_ID(doc2)) = obs_Ancestor(doc1))
            then
              doc2 ∈ c
            else
              doc2 ∉ c
            end)

type
  EventType == Create
             | Copy
             | Edit
             | RemoveDoc
             | Export
             | GetFromDir
             | PutInDir
             | Send,

  Event ::
    evt_type         : EventType
    evt_executedby   : PersonID
    evt_time         : Time
    evt_place        : PlaceID
    evt_exportid     : ExportID
    evt_exportloc    : LocationID
    evt_dossierid    : DossierID
    evt_sendtoperson : PersonID
    evt_sendtoplace  : PlaceID

type
  CommandName == Copy
               | Edit
               | RemoveDoc
               | RemoveDos
               | GetDocFromDos
               | PutDocInDos
               | GetDosFromDos
               | PutDosInDos
               | GetDocFromDir
               | PutDocInDir
               | GetDosFromDir
               | PutDosInDir
               | ExportDoc
               | SignDocument
               | ResetDocMembership
               | ResetDosMembership
               | SendDoc
               | SendDos
               | SetDocPermission
               | SetDosPermission,

  CommandLocks = CommandName →m Keys

value
  CreatePerson: System × Place × Person × PersonID → System,
```

```
      DeletePerson: System × Person → System,
      CreatePlace: System × PlaceID → System,
      DeletePlace: System × PlaceID → System,

      CreateKey: System × Place → System,
      DeleteKey: System × Place × Key → System,
      AssignKey: System × Place × Person × Key → System,
      RemoveKey: System × Place × Person × Key → System,
      CreateDirIndex: System × Place × Index* → System,
      DeleteDirIndex: System × Place × Index* → System,
      CreateLocation: System × Place × Location → System,
      DeleteLocation: System × Place × Location → System,
      RestoreDocument: System × Place × Person × DocumentID → System,

      ManipulateSystemState: System → System
  end
```

E.2 edocsysbasics.rsl

```
edocsystypes
scheme eDocSysBasics =
  extend eDocSysTypes with
  class
    value
      ∪ : RecycleBin × Document-set → RecycleBin
      bin ∪ ds ≡
        let b:RecycleBin •
          obs_Documents(b) = obs_Documents(bin) ∪ ds ∧
          obs_Dossiers(b) = obs_Dossiers(bin)
        in b end,

      ∪ : RecycleBin × Dossier-set → RecycleBin
      bin ∪ ds ≡
        let b:RecycleBin •
          obs_Documents(b) = obs_Documents(bin) ∧
          obs_Dossiers(b) = obs_Dossiers(bin) ∪ ds
        in b end,

      ∪ : Dossier × Document-set → Dossier
      dos ∪ ds ≡
        let d:Dossier •
          obs_ID(d) = obs_ID(dos) ∧
          obs_Description(d) = obs_Description(dos) ∧
          obs_DirMembership(d) = obs_DirMembership(dos) ∧
          obs_DossierMembership(d) = obs_DossierMembership(dos) ∧
          obs_CommandLocks(d) = obs_CommandLocks(dos) ∧
          obs_PlaceID(d) = obs_PlaceID(dos) ∧
          obs_Documents(d) = obs_Documents(dos) ∪ ds ∧
          obs_Dossiers(d) = obs_Dossiers(dos)
        in d end,

      \ : Dossier × Document-set → Dossier
      dos \ ds ≡
        let d:Dossier •
          obs_ID(d) = obs_ID(dos) ∧
          obs_Description(d) = obs_Description(dos) ∧
          obs_DirMembership(d) = obs_DirMembership(dos) ∧
          obs_DossierMembership(d) = obs_DossierMembership(dos) ∧
          obs_CommandLocks(d) = obs_CommandLocks(dos) ∧
          obs_PlaceID(d) = obs_PlaceID(dos) ∧
          obs_Documents(d) = obs_Documents(dos) \ ds ∧
```

```
      obs_Dossiers(d) = obs_Dossiers(dos)
    in d end,

  ∪ : Dossier × Dossier-set → Dossier
  dos ∪ doss ≡
    let d:Dossier •
      obs_ID(d) = obs_ID(dos) ∧
      obs_Description(d) = obs_Description(dos) ∧
      obs_DirMembership(d) = obs_DirMembership(dos) ∧
      obs_DossierMembership(d) = obs_DossierMembership(dos) ∧
      obs_CommandLocks(d) = obs_CommandLocks(dos) ∧
      obs_PlaceID(d) = obs_PlaceID(dos) ∧
      obs_Documents(d) = obs_Documents(dos) ∧
      obs_Dossiers(d) = obs_Dossiers(dos) ∪ doss
    in d end,

  \ : Dossier × Dossier-set → Dossier
  dos \ doss ≡
    let d:Dossier •
      obs_ID(d) = obs_ID(dos) ∧
      obs_Description(d) = obs_Description(dos) ∧
      obs_DirMembership(d) = obs_DirMembership(dos) ∧
      obs_DossierMembership(d) = obs_DossierMembership(dos) ∧
      obs_CommandLocks(d) = obs_CommandLocks(dos) ∧
      obs_PlaceID(d) = obs_PlaceID(dos) ∧
      obs_Documents(d) = obs_Documents(dos) ∧
      obs_Dossiers(d) = obs_Dossiers(dos) \ doss
    in d end,

  ∪ : PersonDossier × Document-set → Person
  perdos ∪ docs ≡
    let (per,doslist) = perdos in
      if len doslist = 0 then
        per ∪ docs
      else
        (per \ {hd(doslist)}) ∪ {doslist ∪ docs}
      end
    end,

  ∪ : PersonDossier × Dossier-set → Person
  perdos ∪ doss ≡
    let (per,doslist) = perdos in
      if len doslist = 0 then
        per ∪ doss
      else
        (per \ {hd(doslist)}) ∪ {doslist ∪ doss}
      end
    end,

  \ : PersonDossier × Document-set → Person
  perdos \ docs ≡
    let (per,doslist) = perdos in
      if len doslist = 0 then
        per \ docs
      else
        (per \ {hd(doslist)}) ∪ {doslist \ docs}
      end
    end,

  \ : PersonDossier × Dossier-set → Person
  perdos \ doss ≡
    let (per,doslist) = perdos in
```

```
        if len doslist = 0 then
          per \ doss
        else
          (per \ {hd(doslist)}) ∪ {doslist \ doss}
        end
      end,

    ∪ : Dossier* × Document-set → Dossier
    doslist ∪ docs ≡
      if len doslist = 1 then
        hd(doslist) ∪ docs
      else
        (hd(doslist) \ {hd(tl(doslist))}) ∪ {tl(doslist) ∪ docs}
      end,

    ∪ : Dossier* × Dossier-set → Dossier
    doslist ∪ doss ≡
      if len doslist = 1 then
        hd(doslist) ∪ doss
      else
        (hd(doslist) \ {hd(tl(doslist))}) ∪ {tl(doslist) ∪ doss}
      end,

    \ : Dossier* × Document-set → Dossier
    doslist \ docs ≡
      if len doslist = 1 then
        hd(doslist) \ docs
      else
        (hd(doslist) \ {hd(tl(doslist))}) ∪ {tl(doslist) \ docs}
      end,

    \ : Dossier* × Dossier-set → Dossier
    doslist \ doss ≡
      if len doslist = 1 then
        hd(doslist) \ doss
      else
        (hd(doslist) \ {hd(tl(doslist))}) ∪ {tl(doslist) \ doss}
      end

  value
    addSignature : Document × Signature → Document
    addSignature(doc, sign) ≡
      let d:Document •
        obs_ID(d) = obs_ID(doc) ∧
        obs_Time(d) = obs_Time(doc) ∧
        obs_PlaceID(d) = obs_PlaceID(doc) ∧
        obs_Contents(d) = obs_Contents(doc) ∧
        obs_Type(d) = obs_Type(doc) ∧
        obs_Creator(d) = obs_Creator(doc) ∧
        obs_Signatures(d) = obs_Signatures(doc) ∪ {sign} ∧
        obs_DirMembership(d) = obs_DirMembership(doc) ∧
        obs_DossierMembership(d) = obs_DossierMembership(doc) ∧
        obs_Ancestor(d) = obs_Ancestor(doc) ∧
        obs_Events(d) = obs_Events(doc) ∧
        obs_CommandLocks(d) = obs_CommandLocks(doc)
      in d end,

    setMembership : Document × DirPath × DossierMembership → Document
    setMembership(doc, dirmem, dosmem) ≡
      let d:Document •
        obs_ID(d) = obs_ID(doc) ∧
        obs_Time(d) = obs_Time(doc) ∧
```

```
      obs_PlaceID(d) = obs_PlaceID(doc) ∧
      obs_Contents(d) = obs_Contents(doc) ∧
      obs_Type(d) = obs_Type(doc) ∧
      obs_Creator(d) = obs_Creator(doc) ∧
      obs_Signatures(d) = obs_Signatures(doc) ∧
      obs_DirMembership(d) = dirmem ∧
      obs_DossierMembership(d) = dosmem ∧
      obs_Ancestor(d) = obs_Ancestor(doc) ∧
      obs_Events(d) = obs_Events(doc) ∧
      obs_CommandLocks(d) = obs_CommandLocks(doc)
    in
      d
    end,

  setMembership : Document-set × DirPath × DossierMembership → Document-set
  setMembership(docs, dirmem, dosmem) as d
    post (all doc:Document • doc ∈ docs ⇒
      setMembership(doc, dirmem, dosmem) ∈ d),

  setMembership : Dossier × DirPath × DossierMembership → Dossier
  setMembership(dos, dirmem, dosmem) ≡
    let d:Dossier •
      obs_ID(d) = obs_ID(dos) ∧
      obs_Description(d) = obs_Description(dos) ∧
      obs_DirMembership(d) = dirmem ∧
      obs_DossierMembership(d) = dosmem ∧
      obs_CommandLocks(d) = obs_CommandLocks(dos) ∧
      obs_PlaceID(d) = obs_PlaceID(dos) ∧
      obs_Documents(d) = obs_Documents(dos) ∧
      obs_Dossiers(d) = obs_Dossiers(dos)
    in
      dos
    end,

  addEvent : Document × Event → Document
  addEvent(document,evt) ≡
    let doc:Document •
      obs_ID(doc) = obs_ID(document) ∧
      obs_Time(doc) = obs_Time(document) ∧
      obs_Contents(doc) = obs_Contents(document) ∧
      obs_Type(doc) = obs_Type(document) ∧
      obs_Creator(doc) = obs_Creator(document) ∧
      obs_PlaceID(doc) = obs_PlaceID(document) ∧
      obs_Ancestor(doc) = obs_Ancestor(document) ∧
      obs_Signatures(doc) = obs_Signatures(document) ∧
      obs_DirMembership(doc) = obs_DirMembership(document) ∧
      obs_DossierMembership(doc) = obs_DossierMembership(document) ∧
      obs_Events(doc) = obs_Events(document) ⁀ ⟨evt⟩ ∧
      obs_CommandLocks(doc) = obs_CommandLocks(document)
    in
      doc
    end,

  addEvent : Document-set × Event → Document-set
  addEvent(docs, evt) as d
    post (all doc:Document • doc ∈ docs ⇒
      addEvent(doc,evt) ∈ d),

  addEvent : Dossier × Event → Dossier
  addEvent(dos, evt) as d
    post (all doc:Document • doc ∈ dos ⇒
      addEvent(doc,evt) ∈ d),
```

```
dossierListIsValid : Dossier* → Bool
dossierListIsValid(doslist) ≡
  if len doslist = 0 ∨ len doslist = 1 then
    true
  elsif hd(tl(doslist)) ∉ obs_Dossiers(hd(doslist)) then
    false
  else
    dossierListIsValid(tl(doslist))
  end,

mostRecentVersion : Document × Document-set → Bool
mostRecentVersion(doc,docs) ≡
  doc = lastVersion(docs),

copiedFrom : Document-set × DocumentID-set → Document
copiedFrom(docs,docids) ≡
  assert(∃! doc:Document •
    obs_ID(doc) ∈ docids ∧
    mk_did(obs_ID(doc)) = obs_Ancestor(firstVersion(docs)));
  let doc:Document •
    obs_ID(doc) ∈ docids ∧
    mk_did(obs_ID(doc)) = obs_Ancestor(firstVersion(docs))
  in
    doc
  end,

firstVersion : Document-set → Document
firstVersion(docs) as d
  post
    d ∈ docs ∧
    ∼(∃ doc:Document •
      doc ∈ docs ∧
      mk_did(obs_ID(doc)) = obs_Ancestor(d)),

lastVersion : Document-set → Document
lastVersion(docs) as d
  post
    d ∈ docs ∧
    ∼(∃ doc:Document •
      doc ∈ docs ∧
      mk_did(obs_ID(d)) = obs_Ancestor(doc)),

setPermission : Document × Keys × CommandName → Document
setPermission(doc,keys,cmd) ≡
  let d:Document •
    obs_ID(d) = obs_ID(doc) ∧
    obs_Time(d) = obs_Time(doc) ∧
    obs_Contents(d) = obs_Contents(doc) ∧
    obs_Type(d) = obs_Type(doc) ∧
    obs_Creator(d) = obs_Creator(doc) ∧
    obs_PlaceID(d) = obs_PlaceID(doc) ∧
    obs_Ancestor(d) = obs_Ancestor(doc) ∧
    obs_Signatures(d) = obs_Signatures(doc) ∧
    obs_DirMembership(d) = obs_DirMembership(doc) ∧
    obs_DossierMembership(d) = obs_DossierMembership(doc) ∧
    obs_Events(d) = obs_Events(doc) ∧
    if keys = {} then
      obs_CommandLocks(d) = obs_CommandLocks(doc) \ {cmd}
    else
      obs_CommandLocks(d) = obs_CommandLocks(doc) † [cmd ↦ keys]
    end
```

```
        in
          doc
        end,

      setPermission : Document-set × Keys × CommandName → Document-set
      setPermission(docs,keys,cmd) as d
        post (all doc:Document •
          doc ∈ docs ⇒ setPermission(doc,keys,cmd) ∈ d),

      setPermission : Dossier × Keys × CommandName → Dossier
      setPermission(dos,keys,cmd) ≡
        let d:Dossier •
          obs_ID(d) = obs_ID(dos) ∧
          obs_Description(d) = obs_Description(dos) ∧
          obs_DirMembership(d) = obs_DirMembership(dos) ∧
          obs_DossierMembership(d) = obs_DossierMembership(dos) ∧
          obs_PlaceID(d) = obs_PlaceID(dos) ∧
          obs_Documents(d) = obs_Documents(dos) ∧
          obs_Dossiers(d) = obs_Dossiers(dos) ∧
          if keys = {} then
            obs_CommandLocks(d) = obs_CommandLocks(dos) \ {cmd}
          else
            obs_CommandLocks(d) = obs_CommandLocks(dos) † [cmd ↦ keys]
          end
        in
          dos
        end,

      hasPermission : Person × Document × CommandName → Bool
      hasPermission(person,doc,cmd) ≡
        let doclocks = obs_CommandLocks(doc) in
          if cmd ∈ dom doclocks then
            doclocks(cmd) ⊂ obs_Keys(person)
          else
            true
          end
        end,

      hasPermission : Person × Dossier × CommandName → Bool
      hasPermission(person,dos,cmd) ≡
        let doslocks = obs_CommandLocks(dos) in
          if cmd ∈ dom doslocks then
            doslocks(cmd) ⊂ obs_Keys(person)
          else
            true
          end
        end
  end
```

E.3 edocsyswf.rsl

```
edocsysBasics
scheme eDocSysWF =
  extend eDocSysBasics with
  class
    value
      wf_doc : System′ → Bool
      wf_doc((places,docids_in_use,dosids_in_use,cpyids_in_use)) ≡ (
        ∀ doc,doc2:Document •
          obs_ID(doc) ∈ docids_in_use ⇒ (
          obs_ID(doc) = obs_ID(doc2) ⇒ doc = doc2 ∧
```

```
        obs_DirMembership(doc) ≠ none ⇒ obs_DossierMembership(doc) = none ∧
        obs_DossierMembership(doc) ≠ none ⇒ obs_DirMembership(doc) = none ∧
        obs_Type(doc) = master ⇒ obs_Ancestor(doc) = none ∧
        obs_Type(doc) ≠ master ⇒ obs_Ancestor(doc) ≠ none ∧
        (∃! (dir,pers,locs,bin,keys):Place, person:Person, loc:Location •
          (dir,pers,locs,bin,keys) ∈ rng places ∧
          (Xor(Xor(Xor(person ∈ rng pers ∧ doc ∈ person,
                      loc ∈ rng locs ∧ doc ∈ loc),
                      doc ∈ dir),
                      doc ∈ obs_Documents(bin) )))))

    value
      wf_dos : System′ → Bool
      wf_dos((places,docids_in_use,dosids_in_use,cpyids_in_use)) ≡ (
        ∀ dos,dos2:Dossier •
          obs_ID(dos) ∈ dosids_in_use ∧
          obs_DirMembership(dos) ≠ none ⇒ obs_DossierMembership(dos) = none ∧
          obs_DossierMembership(dos) ≠ none ⇒ obs_DirMembership(dos) = none ∧
          (obs_ID(dos) = obs_ID(dos2) ⇒ dos = dos2) ∧
          (∃! (dir,pers,locs,bin,keys):Place, person:Person, loc:Location •
            (dir,pers,locs,bin,keys) ∈ rng places ∧
            (Xor(Xor(Xor(person ∈ rng pers ∧ dos ∈ person,
                        loc ∈ rng locs ∧ dos ∈ loc),
                        dos ∈ dir),
                        dos ∈ obs_Dossiers(bin)))))

    value
      wf_pers : System′ → Bool
      wf_pers((places,docids_in_use,dosids_in_use,cpyids_in_use)) ≡ (
        ∀ pers,pers2:Person •
          (obs_ID(pers) = obs_ID(pers2) ⇒ pers = pers2) ∧
          (obs_Signature(pers) = obs_Signature(pers2) ⇒ pers = pers2) ∧
          (∃ (dir,pers1,locs,bin,keys):Place •
            (dir,pers1,locs,bin,keys) ∈ rng places ∧ pers ∈ rng pers1))

    axiom
      ∀ w:System′ •
        wf_system(w) ≡ (wf_doc(w) ∧ wf_dos(w) ∧ wf_pers(w))


  end
```

E.4 edocsyscmds.rsl

```
edocsyswf
scheme eDocSysCmds =
  extend eDocSysWF with
  class
    type
      Command = CreateDoc
              | CreateDos
              | Copy
              | Edit
              | RemoveDoc
              | RemoveDos
              | GetDocFromDos
              | PutDocInDos
              | GetDosFromDos
              | PutDosInDos
              | GetDocFromDir
              | PutDocInDir
```

```
            | GetDosFromDir
            | PutDosInDir
            | ExportDoc
            | SignDocument
            | ResetDocMembership
            | ResetDosMembership
            | SendDoc
            | SendDos
            | SetDocPermission
            | SetDosPermission
            | ReturnDoc
            | ReturnDos
            | Merge

type
  CreateDoc ::
    ref_person : Person
    ref_PlaceID : PlaceID
    ref_time : Time
    ref_contents : Contents,

  CreateDos ::
    ref_person : Person
    ref_PlaceID : PlaceID
    ref_time : Time
    ref_desc    : DossierDescription,

  Copy ::
    ref_person : Person
    ref_PlaceID : PlaceID
    ref_time : Time
    ref_doc : Document,

  Edit ::
    ref_person : Person
    ref_PlaceID : PlaceID
    ref_time : Time
    ref_doc : Document
    ref_edition : FTE,

  RemoveDoc ::
    ref_person : Person
    ref_PlaceID : PlaceID
    ref_time : Time
    ref_doc : Document,

  RemoveDos ::
    ref_person : Person
    ref_PlaceID : PlaceID
    ref_time : Time
    ref_doc : Dossier,

  GetDocFromDos ::
    ref_person : Person
    ref_PlaceID : PlaceID
    ref_time : Time
    ref_dos : Dossier
    ref_doc : Document,

  PutDocInDos ::
    ref_person : Person
    ref_PlaceID : PlaceID
```

```
      ref_time : Time
      ref_dos : Dossier
      ref_doc : Document,

    GetDosFromDos ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_dos : Dossier
      ref_doc : Dossier,

    PutDosInDos ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_dos : Dossier
      ref_doc : Dossier,

    GetDocFromDir ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_dirpath : Index*
      ref_docid : DocumentID,

    PutDocInDir ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_dirpath : Index*
      ref_doc : Document,

    GetDosFromDir ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_dirpath : Index*
      ref_dosid : DossierID,

    PutDosInDir ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_dirpath : Index*
      ref_dos : Dossier,

    ExportDoc ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_locid : LocationID
      ref_doc : Document,

    SignDocument ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_doc : Document
      ref_sign : Signature,

    SendDoc ::
      ref_sender : Person
```

```
      ref_origin : PlaceID
      ref_time : Time
      ref_reciever : PersonID
      ref_dest : PlaceID
      ref_doc : Document,

    SendDos ::
      ref_sender : Person
      ref_origin : PlaceID
      ref_time : Time
      ref_reciever : PersonID
      ref_dest : PlaceID
      ref_doc : Dossier,

    ResetDocMembership ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_doc : Document,

    ResetDosMembership ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_dos : Dossier,

    SetDocPermission ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_doc : Document
      ref_keys : Keys
      ref_cmd : CommandName,

    SetDosPermission ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_dos : Dossier
      ref_keys : Keys
      ref_cmd : CommandName,

    ReturnDoc ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_doc : Document,

    ReturnDos ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_dos : Dossier,

    Merge ::
      ref_person : Person
      ref_PlaceID : PlaceID
      ref_time : Time
      ref_doc : Document

  value
    M: Command → System → System
```

```
M(cmd)(places, docids, dosids, copyids) ≡
  case cmd of
    mk_CreateDoc(person, plid, time, contents) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        assert(person ∈ rng pers);
        let did:DocumentID • did ∉ docids in
          let doc:Document •
            obs_ID(doc) = did ∧
            obs_Time(doc) = time ∧
            obs_Contents(doc) = contents ∧
            obs_Type(doc) = master ∧
            obs_Creator(doc) = obs_ID(person) ∧
            obs_PlaceID(doc) = plid ∧
            obs_Ancestor(doc) = none ∧
            obs_Signatures(doc) = {} ∧
            obs_DirMembership(doc) = none ∧
            obs_DossierMembership(doc) = none ∧
            obs_CommandLocks(doc) = [] ∧
            obs_Events(doc) = ⟨⟩
          in
            let evt:Event •
              evt_type(evt) = Create ∧
              evt_executedby(evt) = obs_ID(person) ∧
              evt_time(evt) = time ∧
              evt_place(evt) = plid
            in
              (places † [plid ↦
                (dir, pers † [obs_ID(person) ↦ person ∪ {addEvent(doc,evt
)}],
                locs,bin,keys)], docids ∪ {did}, dosids,copyids)
            end
          end
        end
      end,

    mk_CreateDos(person, plid, time, desc) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        assert(person ∈ rng pers);
        let dosid:DossierID • dosid ∉ dosids in
          let dos:Dossier •
            obs_ID(dos) = dosid ∧
            obs_Description(dos) = desc ∧
            obs_CommandLocks(dos) = [] ∧
            obs_DirMembership(dos) = none ∧
            obs_DossierMembership(dos) = none ∧
            obs_PlaceID(dos) = plid ∧
            obs_Documents(dos) = {} ∧
            obs_Dossiers(dos) = {}
          in
            (places † [plid ↦
              (dir, pers † [obs_ID(person) ↦ person ∪ {dos}],
              locs,bin,keys)], docids, dosids ∪ {dosid}, copyids)
          end
        end
      end,

    mk_Copy(person, plid, time, doc) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        let docs = obs_Group(doc,docids) in
          assert(hasPermission(person,doc,Copy) ∧
                 person ∈ rng pers ∧
                 docs ⊂ obs_Documents(person));
```

```
          let did:DocumentID • did ∉ docids in
            let cpy:Document •
              obs_ID(cpy) = did ∧
              obs_Time(cpy) = time ∧
              obs_Contents(cpy) = obs_Contents(doc) ∧
              obs_Type(cpy) = copy ∧
              obs_Creator(cpy) = obs_ID(person) ∧
              obs_PlaceID(cpy) = plid ∧
              obs_Ancestor(cpy) = mk_did(obs_ID(doc)) ∧
              obs_Signatures(cpy) = obs_Signatures(doc) ∧
              obs_DirMembership(cpy) = none ∧
              obs_DossierMembership(cpy) = none ∧
              obs_CommandLocks(doc) = [] ∧
              obs_Events(cpy) = ⟨⟩
            in
              let evt:Event •
                evt_type(evt) = Copy ∧
                evt_executedby(evt) = obs_ID(person) ∧
                evt_time(evt) = time ∧
                evt_place(evt) = plid
              in
                (places † [plid ↦
                  (dir, pers † [obs_ID(person) ↦
                    ((person \ {doc}) ∪ {addEvent(doc,evt)}) ∪ {addEvent(
cpy,evt)}],
                  locs,bin,keys)], docids ∪ {did}, dosids,copyids)
              end
            end
          end
        end
      end,

    mk_Edit(person, plid, time, document, (te,fe)) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        let docs = obs_Group(document,docids) in
          assert(hasPermission(person,document,Edit) ∧
                 person ∈ rng pers ∧
                 mostRecentVersion(document,docs) ∧
                 docs ⊂ obs_Documents(person));
          let docid:DocumentID • docid ∉ docids in
            let doc:Document •
              obs_ID(doc) = docid ∧
              obs_Time(doc) = time ∧
              obs_Contents(doc) = te(obs_Contents(document)) ∧
              obs_Type(doc) = version ∧
              obs_Creator(doc) = obs_ID(person) ∧
              obs_PlaceID(doc) = plid ∧
              obs_Ancestor(doc) = mk_did(obs_ID(document)) ∧
              obs_Signatures(doc) = {} ∧
              obs_DirMembership(doc) = obs_DirMembership(document) ∧
              obs_DossierMembership(doc) = obs_DossierMembership(document
) ∧
              obs_CommandLocks(doc) = obs_CommandLocks(document) ∧
              obs_Events(doc) = obs_Events(document)
            in
              let evt:Event •
                evt_type(evt) = Edit ∧
                evt_executedby(evt) = obs_ID(person) ∧
                evt_time(evt) = time ∧
                evt_place(evt) = plid
              in
              (places † [plid ↦
```

```
                (dir, pers † [obs_ID(person) ↦
                  ((person \ {document}) ∪
                    {addEvent(document,evt)}) ∪
                      {addEvent(doc,evt)}],
                locs,bin,keys)], docids ∪ {docid}, dosids, copyids)
              end
            end
          end
        end
      end,

    mk_RemoveDoc(person, plid, time, doc) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        let docs = obs_Group(doc,docids) in
          assert(hasPermission(person,doc,RemoveDoc) ∧
                 person ∈ rng pers ∧
                 docs ⊂ obs_Documents(person));
          let evt:Event •
            evt_type(evt) = RemoveDoc ∧
            evt_executedby(evt) = obs_ID(person) ∧
            evt_time(evt) = time ∧
            evt_place(evt) = plid
          in
            (places † [plid ↦
              (dir, pers † [obs_ID(person) ↦ person \ docs],
              locs,bin ∪ addEvent(docs,evt),keys)], docids, dosids,
copyids)
          end
        end
      end,

    mk_RemoveDos(person, plid, time, dos) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        assert(hasPermission(person,dos,RemoveDos) ∧
               person ∈ rng pers ∧
               dos ∈ obs_Dossiers(person) ∧
               obs_Documents(dos) = {} ∧
               obs_Dossiers(dos) = {});
        (places † [plid ↦
          (dir, pers † [obs_ID(person) ↦ person \ {dos}],
          locs,bin ∪ {dos},keys)], docids, dosids, copyids)
       end,

    mk_GetDocFromDos(person, plid, time, dos, doc) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        let docs = obs_Group(doc,docids) in
          assert(hasPermission(person,doc,GetDocFromDos) ∧
                 person ∈ rng pers ∧
                 docs ⊂ obs_Documents(dos) ∧
                 dos ∈ obs_Dossiers(person));
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              ((person \ {dos}) ∪ {dos \ docs}) ∪
                docs], locs,bin,keys)],
            docids, dosids, copyids)
        end
      end,

    mk_PutDocInDos(person, plid, time, dos, doc) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        let docs = obs_Group(doc,docids) in
          assert(hasPermission(person,doc,PutDocInDos) ∧
```

```
                  obs_DirMembership(doc) = none ∧
                    (obs_DossierMembership(doc) = mk_did(obs_ID(dos)) ∨
                     obs_DossierMembership(doc) = none) ∧
                  person ∈ rng pers ∧
                  docs ⊂ obs_Documents(person) ∧
                  dos ∈ obs_Dossiers(person));
            (places † [plid ↦
              (dir, pers † [obs_ID(person) ↦ ((person \ docs) \ {dos}) ∪
                  {dos ∪ setMembership(docs,none,mk_did(obs_ID(dos)))}],
locs, bin,keys)],
              docids, dosids, copyids)
          end
        end,

      mk_GetDosFromDos(person, plid, time, dos, dos1) →
        let (dir,pers,locs,bin,keys) = places(plid) in
          assert(hasPermission(person,dos,GetDosFromDos) ∧
                 person ∈ rng pers ∧
                 dos1 ∈ dos ∧
                 dos ∈ obs_Dossiers(person));
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              ((person \ {dos}) ∪ {dos \ {dos1}}) ∪
                {dos1}], locs,bin,keys)],
            docids, dosids, copyids)
        end,

      mk_PutDosInDos(person, plid, time, dos, dos1) →
        let (dir,pers,locs,bin,keys) = places(plid) in
          assert(hasPermission(person,dos,PutDosInDos) ∧
                 obs_DirMembership(dos1) = none ∧
                   (obs_DossierMembership(dos1) = mk_did(obs_ID(dos)) ∨
                    obs_DossierMembership(dos1) = none) ∧
                 person ∈ rng pers ∧
                 dos1 ∈ obs_Dossiers(person) ∧
                 dos ∈ obs_Dossiers(person));
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              ((person \ {dos1}) \ {dos}) ∪
                {dos ∪ {setMembership(dos1,none,mk_did(obs_ID(dos)))}}],
locs,bin,keys)],
            docids, dosids, copyids)
        end,

      mk_GetDocFromDir(person, plid, time, dirpath, docid) →
        let (dir,pers,locs,bin,keys) = places(plid) in
          let doc:Document • obs_ID(doc) = docid in
            let docs = obs_Group(doc,docids) in
              assert(hasPermission(person,doc,GetDocFromDir) ∧
                     person ∈ rng pers ∧
                     indexExists(dir,dirpath) ∧
                     docs ⊂ obs_Documents(recurseDir(dir,dirpath)) ∧
                     obs_Keys(recurseDir(dir,dirpath)) ⊂ obs_Keys(person));
              let evt:Event •
                evt_type(evt) = GetFromDir ∧
                evt_executedby(evt) = obs_ID(person) ∧
                evt_time(evt) = time ∧
                evt_place(evt) = plid
              in
                (places † [plid ↦
                  (updateDir(dir,dirpath,recurseDir(dir,dirpath) \
                     docs),
```

```
                    pers † [obs_ID(person) ↦
                      (person ∪ addEvent(docs,evt))],locs,bin,keys)],
                    docids, dosids, copyids)
                end
              end
            end
          end,

        mk_PutDocInDir(person, plid, time, dirpath, doc) →
          let (dir,pers,locs,bin,keys) = places(plid) in
            let docs = obs_Group(doc,docids) in
              assert(hasPermission(person,doc,PutDocInDir) ∧
                     obs_DossierMembership(doc) = none ∧
                       (obs_DirMembership(doc) = mk_dip(dirpath) ∨
                        obs_DirMembership(doc) = none) ∧
                     person ∈ rng pers ∧
                     docs ⊂ obs_Documents(person) ∧
                     obs_Keys(recurseDir(dir,dirpath)) ⊂ obs_Keys(person));
              let evt:Event •
                evt_type(evt) = PutInDir ∧
                evt_executedby(evt) = obs_ID(person) ∧
                evt_time(evt) = time ∧
                evt_place(evt) = plid
              in
                (places † [plid ↦
                  (updateDir(dir,dirpath,recurseDir(dir,dirpath) ∪
                    setMembership(addEvent(docs,evt),mk_dip(dirpath),none)),
                   pers † [obs_ID(person) ↦
                     (person \ docs)],locs,bin,keys)],
                   docids, dosids, copyids)
              end
            end
          end,

        mk_GetDosFromDir(person, plid, time, dirpath, dosid) →
          let (dir,pers,locs,bin,keys) = places(plid) in
            let dos:Dossier • obs_ID(dos) = dosid in
              assert(hasPermission(person,dos,GetDosFromDir) ∧
                     person ∈ rng pers ∧
                     indexExists(dir,dirpath) ∧
                     dos ∈ recurseDir(dir,dirpath) ∧
                     obs_Keys(recurseDir(dir,dirpath)) ⊂ obs_Keys(person));
              let evt:Event •
                evt_type(evt) = GetFromDir ∧
                evt_executedby(evt) = obs_ID(person) ∧
                evt_time(evt) = time ∧
                evt_place(evt) = plid
              in
                (places † [plid ↦
                  (updateDir(dir,dirpath,recurseDir(dir,dirpath) \
                     {dos}),
                   pers † [obs_ID(person) ↦
                     (person ∪ {addEvent(dos,evt)})],locs,bin,keys)],
                   docids, dosids, copyids)
              end
            end
          end,

        mk_PutDosInDir(person, plid, time, dirpath, dos) →
          let (dir,pers,locs,bin,keys) = places(plid) in
            assert(hasPermission(person,dos,PutDosInDir) ∧
                   person ∈ rng pers ∧
```

```
                  obs_DossierMembership(dos) = none ∧
                    (obs_DirMembership(dos) = mk_dip(dirpath) ∨
                     obs_DirMembership(dos) = none) ∧
                   dos ∈ obs_Dossiers(person) ∧
                  obs_Keys(recurseDir(dir,dirpath)) ⊂ obs_Keys(person));
          let evt:Event •
            evt_type(evt) = PutInDir ∧
            evt_executedby(evt) = obs_ID(person) ∧
            evt_time(evt) = time ∧
            evt_place(evt) = plid
          in
            (places † [plid ↦
              (updateDir(dir,dirpath,recurseDir(dir,dirpath) ∪
                {setMembership(addEvent(dos,evt),mk_dip(dirpath),none)}),
               pers † [obs_ID(person) ↦
                 (person \ {dos})],locs,bin,keys)],
               docids, dosids, copyids)
          end
        end,

      mk_ExportDoc(person, plid, time, locid, doc) →
        let (dir,pers,locs,bin,keys) = places(plid) in
          let docs = obs_Group(doc,docids) in
            assert(hasPermission(person,doc,ExportDoc) ∧
                   person ∈ rng pers ∧
                   locid ∈ locs ∧
                   docs ⊂ obs_Documents(person));

            /* The actual exporting (printing/cdburning) should be
               done elsewhere */

            let exportid:ExportID • exportid ∉ copyids in
              let evt:Event •
                evt_type(evt) = Export ∧
                evt_executedby(evt) = obs_ID(person) ∧
                evt_time(evt) = time ∧
                evt_place(evt) = plid ∧
                evt_exportid(evt) = exportid ∧
                evt_exportloc(evt) = locid
              in
                (places † [plid ↦ (dir,pers † [obs_ID(person) ↦
                  (person \ {doc}) ∪ {addEvent(doc,evt)}],locs,bin, keys)],
                 docids, dosids, copyids ∪ {exportid})
              end
            end
          end
        end,

      mk_SignDocument(person, plid, time, doc, sign) →
        let (dir,pers,locs,bin,keys) = places(plid) in
          let docs = obs_Group(doc,docids) in
            assert(hasPermission(person,doc,SignDocument) ∧
                   person ∈ rng pers ∧
                   mostRecentVersion(doc,docs) ∧
                   docs ⊂ obs_Documents(person));
              (places † [plid ↦
                (dir, pers † [obs_ID(person) ↦
                  ((person \ {doc}) ∪ {addSignature(doc,sign)})],
                 locs,bin,keys)],
               docids, dosids, copyids)
          end
        end,
```

```
      mk_SendDoc(person, plid_org, time, pid_dst, plid_dst, doc) →
        let (dir,pers,locs,bin,keys) = places(plid_org),
          (dir_dst,pers_dst,locs_dst,bin_dst,keys_dst) = places(plid_dst),
          person_dst : Person • obs_ID(person_dst) = pid_dst
        in
          let docs = obs_Group(doc,docids) in
            assert(hasPermission(person,doc,SendDoc) ∧
                   person ∈ rng pers ∧
                   docs ⊂ obs_Documents(person) ∧
                   pid_dst ∈ dom pers_dst);
            let evt:Event •
              evt_type(evt) = Send ∧
              evt_executedby(evt) = obs_ID(person) ∧
              evt_time(evt) = time ∧
              evt_place(evt) = plid_org ∧
              evt_sendtoperson(evt) = pid_dst ∧
              evt_sendtoplace(evt) = plid_dst
            in
              (places † [plid_org ↦
                (dir, pers † [obs_ID(person) ↦
                  (person \ docs)],
                locs,bin,keys),
               plid_dst ↦
                (dir_dst, pers_dst † [ pid_dst ↦
                  (person_dst ∪ addEvent(docs,evt))],
                 locs_dst,bin_dst,keys_dst)],
              docids, dosids, copyids)
            end
          end
        end,

      mk_SendDos(person, plid_org, time, pid_dst, plid_dst, dos) →
        let (dir,pers,locs,bin,keys) = places(plid_org),
          (dir_dst,pers_dst,locs_dst,bin_dst,keys_dst) = places(plid_dst),
          person_dst : Person • obs_ID(person_dst) = pid_dst
        in
          assert(hasPermission(person,dos,SendDos) ∧
                 person ∈ rng pers ∧
                 dos ∈ obs_Dossiers(person) ∧
                 pid_dst ∈ dom pers_dst);
          let evt:Event •
            evt_type(evt) = Send ∧
            evt_executedby(evt) = obs_ID(person) ∧
            evt_time(evt) = time ∧
            evt_place(evt) = plid_org ∧
            evt_sendtoperson(evt) = pid_dst ∧
            evt_sendtoplace(evt) = plid_dst
          in
            (places † [plid_org ↦
              (dir, pers † [obs_ID(person) ↦
                (person \ {dos})],
              locs,bin,keys),
             plid_dst ↦
              (dir_dst, pers_dst † [ pid_dst ↦
                (person_dst ∪ {addEvent(dos,evt)})],
               locs_dst,bin_dst,keys_dst)],
            docids, dosids, copyids)
          end
        end,

      mk_ResetDocMembership(person, plid, time, doc) →
```

```
      let (dir,pers,locs,bin,keys) = places(plid) in
        let docs = obs_Group(doc,docids) in
          assert(hasPermission(person,doc,ResetDocMembership) ∧
                 person ∈ rng pers ∧
                 docs ⊂ obs_Documents(person));
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              (person \ docs) ∪
                setMembership(docs,none,none)],
            locs,bin,keys)],
          docids, dosids, copyids)
        end
      end,

    mk_ResetDosMembership(person, plid, time, dos) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        assert(hasPermission(person,dos,ResetDosMembership) ∧
               person ∈ rng pers ∧
               dos ∈ obs_Dossiers(person));
        (places † [plid ↦
          (dir, pers † [obs_ID(person) ↦
            (person \ {dos}) ∪
              {setMembership(dos,none,none)}],
          locs,bin,keys)],
        docids, dosids, copyids)
      end,

    mk_SetDocPermission(person, plid, time, doc, keys1, command) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        let docs = obs_Group(doc,docids) in
          assert(hasPermission(person,doc,SetDocPermission) ∧
                 person ∈ rng pers ∧
                 keys1 ⊂ obs_Keys(person) ∧
                 docs ⊂ obs_Documents(person));
          if command = ExportDoc ∨ command = Copy then
            (places † [plid ↦
              (dir, pers † [obs_ID(person) ↦
                (person \ {doc}) ∪ {setPermission(doc,keys1,command)}],
              locs,bin,keys)],
            docids,dosids,copyids)
          else
            (places † [plid ↦
              (dir, pers † [obs_ID(person) ↦
                (person \ docs) ∪ setPermission(docs,keys1,command)],
              locs,bin,keys)],
            docids,dosids,copyids)
          end
        end
      end,

    mk_SetDosPermission(person, plid, time, dos, keys1, command) →
      let (dir,pers,locs,bin,keys) = places(plid) in
          assert(hasPermission(person,dos,SetDosPermission) ∧
                 person ∈ rng pers ∧
                 keys1 ⊂ obs_Keys(person) ∧
                 dos ∈ obs_Dossiers(person));
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              (person \ {dos}) ∪ {setPermission(dos,keys1,command)}],
            locs,bin,keys)],
          docids,dosids,copyids)
      end,
```

```
        mk_ReturnDoc(person, plid, time, doc) →
          let (dir,pers,locs,bin,keys) = places(plid) in
            let docs = obs_Group(doc,docids) in
              assert(person ∈ rng pers ∧
                     docs ⊂ obs_Documents(person) ∧
                     (obs_DossierMembership(doc) ≠ none ∨
                      obs_DirMembership(doc) ≠ none));

              /* the document shall be returned to its membership container
 */

              (places,docids,dosids,copyids)
            end
          end,

        mk_ReturnDos(person, plid, time, dos) →
          let (dir,pers,locs,bin,keys) = places(plid) in
              assert(person ∈ rng pers ∧
                     dos ∈ obs_Dossiers(person) ∧
                     (obs_DossierMembership(dos) ≠ none ∨
                      obs_DirMembership(dos) ≠ none));

              /* the dossier shall be returned to its membership container */

              (places, docids,dosids,copyids)
          end,

        mk_Merge(person, plid, time, doc) →
          let (dir,pers,locs,bin,keys) = places(plid) in
            let docs = obs_Group(doc,docids) in
              let targetgroup = obs_Group(copiedFrom(docs,docids),docids) in
                let targetdoc = lastVersion(targetgroup) in
                  let fte = generateMergeFTE(doc,targetdoc) in
                    assert(hasPermission(person,targetdoc,Edit) ∧
                           hasPermission(person,doc,RemoveDoc) ∧
                           person ∈ rng pers ∧
                           docs ⊂ obs_Documents(person) ∧
                           targetgroup ⊂ obs_Documents(person));
                    let newsystem = M(mk_Edit(person,plid,time,doc,fte))
                      (places, docids, dosids, copyids) in
                      M(mk_RemoveDoc(person,plid,time,doc))(newsystem)
                    end
                  end
                end
              end
            end
          end
      end

    type
      WhereAbouts == mk_dip(DirPath) | mk_did(DossierID) | mk_pid(PersonID)

    value
      /* Find document whereabouts from its ID */
      findDoc : System × DocumentID → WhereAbouts,

      /* Find dossier whereabouts from its ID */
      findDos : System × DossierID → WhereAbouts,

      /* Document history function */
      getDocHist : Document → (Document × (PlaceID × Time × PersonID))*
```

```
      getDocHist(doc) ≡
        let (plid,time,pid) =
          (obs_PlaceID(doc),obs_Time(doc),obs_Creator(doc))
        in
          if obs_Type(doc) = master
            then ⟨(doc,(plid,time,pid))⟩
            else
              let doc2:Document • mk_did(obs_ID(doc2)) = obs_Ancestor(doc) in
                getDocHist(doc2) ⌢ ⟨(doc,(plid,time,pid))⟩
              end
            end
          end,

      /* Sample statistic function */
      getDocList_for_PersonID : System × PersonID → Document-set
      getDocList_for_PersonID((places, docids_in_use, dosids_in_use,
     copyids_in_use),pid) ≡ (
        let docset:Document-set •
          (all doc:Document •
            (obs_ID(doc) ∈ docids_in_use ∧ obs_Creator(doc) = pid) ⇒
              doc ∈ docset)
        in
          docset
        end
      ),

      /* Generate a domain specific transfer function for merging */
      generateMergeFTE : Document × Document → FTE
end
```

Appendix F

DocSys – Secure Protocol Architecture

F.1 securesession.rsl

```
scheme SecureSession =
  class
    channel
      ClientAuthenticate : Data,
      ServerAuthenticate : Data,
      LoginRequest : Data,
      LoginAnswer : Bool,
      SessionKey : Key × Signature,
      SecureConnection : Data,
      ClientSend : Data,
      ServerSend : Data

    type
      Key,
      Signature,
      Data,
      UserPublicKeys = Data →m Key,
      UserPasswords = Data →m Data

    value
      server_publickey : Key,
      server_privatekey : Key,
      server_userpublickeys : UserPublicKeys,
      server_userpasswords : UserPasswords,
      client_privatekey : Key,
      client_username : Data,
      client_password : Data,

      Encrypt : Key × Key → Key,
      Decrypt : Key × Key → Key,
      Encrypt : Data × Key → Data,
      Decrypt : Data × Key → Data,

      Sign : Key × Key → Key × Signature,
      VerifySign : (Key × Signature) × Key → Bool

    value
      System : Unit → in any out any Unit
```

```
 System() ≡
   Server(server_privatekey, server_userpublickeys, server_userpasswords)
 ||
   Client (client_privatekey, server_publickey, client_username,
 client_password),

 Client : Key × Key × Data × Data →
   in  ClientAuthenticate,
       ServerAuthenticate,
       LoginAnswer,
       SessionKey,
       ServerSend
   out ClientAuthenticate,
       ServerAuthenticate,
       LoginRequest,
       ClientSend
   Unit
 Client(privatekey, server_publickey, username, password) ≡
   ProcessClientAuthenticate(privatekey)  ||
   ConnectToServer(privatekey, server_publickey, username, password),

 ProcessClientAuthenticate : Key → in ClientAuthenticate out
 ClientAuthenticate Unit
 ProcessClientAuthenticate(privatekey) ≡
   let nonce = ClientAuthenticate ? in
     ClientAuthenticate ! Encrypt(nonce,privatekey);
     ProcessClientAuthenticate(privatekey)
   end,

 ConnectToServer : Key × Key × Data × Data →
   in  ServerAuthenticate,
       LoginAnswer,
       SessionKey,
       ServerSend
   out ServerAuthenticate,
       LoginRequest,
       ClientSend
   Unit
 ConnectToServer(privatekey, server_publickey, username, password) ≡
  /* Server authentication */
   let nonce : Data in
     ServerAuthenticate ! nonce;
     let encrypted_nonce = ServerAuthenticate ? in
       if Encrypt(encrypted_nonce,server_publickey) ≠ nonce then
         chaos
       end
     end
   end;

   /* Login with username */
   LoginRequest ! Encrypt(username, server_publickey);

   /* Await server response */
   let answer = LoginAnswer? in
     if ∼answer then
       chaos
     end
   end;

   /* Receive session key */
   let (session_key,signature) = SessionKey ? in
     if ∼VerifySign((session_key, signature),server_publickey) then
```

```
          chaos
        end;

        /* Await server response */
        let answer = LoginAnswer? in
          if ∼answer then
            chaos
          end
        end;

        /* Send password on secure connection */
        LoginRequest ! Encrypt(password, Encrypt(session_key,
   client_privatekey));

        /* Await server response */
        let answer = LoginAnswer? in
          if ∼answer then
            chaos
          end
        end;

        /* Secure session established */
        SecureSessionClient(Encrypt(session_key, client_privatekey))
      end,

    SecureSessionClient : Key → in ServerSend out ClientSend Unit
    SecureSessionClient(session_key) ≡
      SecureSenderClient(session_key) ∥
      SecureReceiverClient(session_key),

    SecureSenderClient : Key → out ClientSend Unit
    SecureSenderClient(session_key) ≡
      /* Send some data to server */
      let message : Data in
        ClientSend ! Encrypt(message,session_key);
        SecureSenderClient(session_key)
      end,

    SecureReceiverClient : Key → in ServerSend Unit
    SecureReceiverClient(session_key) ≡
      /* Receive some data from server */
      let encrypted_message = ServerSend ? in
        let message = Encrypt(encrypted_message,session_key) in
          /* Process data received */
          SecureReceiverClient(session_key)
        end
      end,

    Server : Key × UserPublicKeys × UserPasswords → in any out any Unit
    Server(privatekey, userpublickeys, userpasswords) ≡
      ProcessServerAuthenticate(privatekey) ∥
      ProcessClientConnection(privatekey, userpublickeys, userpasswords),

    ProcessServerAuthenticate : Key → in ServerAuthenticate out
   ServerAuthenticate Unit
    ProcessServerAuthenticate(privatekey) ≡
      let nonce = ServerAuthenticate ? in
        ServerAuthenticate ! Encrypt(nonce,privatekey);
        ProcessServerAuthenticate(privatekey)
      end,

    ProcessClientConnection : Key × UserPublicKeys × UserPasswords →
```

```
      in  ServerAuthenticate,
          ClientAuthenticate,
          LoginRequest,
          ClientSend
      out ServerAuthenticate,
          ClientAuthenticate,
          LoginAnswer,
          SessionKey,
          ServerSend
      Unit
     ProcessClientConnection(privatekey, userpublickeys, userpasswords) ≡
      /* Receive username */
      let encrypted_username = LoginRequest ? in
        let username = Encrypt(encrypted_username,privatekey) in

          /* User must exist in server user lists */
          if username ∉ dom userpublickeys ∨ username ∉ dom userpasswords then
             LoginAnswer! false;
             ProcessClientConnection(privatekey, userpublickeys, userpasswords)
          end;

          /* Authenticate client */
          let nonce : Data in
            ClientAuthenticate ! nonce;
            let encrypted_nonce = ClientAuthenticate ? in
              if Encrypt(encrypted_nonce,userpublickeys(username)) ≠ nonce then
                LoginAnswer! false;
                ProcessClientConnection(privatekey, userpublickeys, userpasswords)
              end
            end
          end;

          /* User ∃ and ≡ authenticated */
          LoginAnswer! true;

          /* Establish a session key */
          let session_key : Key in
            SessionKey ! Sign(Encrypt(session_key,userpublickeys(username)), privatekey);

            /* Indicate that secure session ≡ ready */
            LoginAnswer! true;

            /* Receive password on secure connection */
            let encrypted_password = LoginRequest ? in
              let password = Decrypt(encrypted_password,session_key) in
                if userpasswords(username) ≠ password then
                  LoginAnswer ! false;
                  ProcessClientConnection(privatekey,userpublickeys, userpasswords)
                end
              end
            end;

            /* Indicate that password was accepted */
            LoginAnswer! true;

            /* Secure session established */
```

```
              SecureSessionServer(session_key)
            end
          end
        end,

      SecureSessionServer : Key → in ClientSend out ServerSend Unit
      SecureSessionServer(session_key) ≡
        SecureSenderServer(session_key) ∥
        SecureReceiverServer(session_key),

      SecureSenderServer : Key → out ServerSend Unit
      SecureSenderServer(session_key) ≡
        /* Send some data to client */
        let message : Data in
          ServerSend ! Encrypt(message,session_key);
          SecureSenderServer(session_key)
        end,

      SecureReceiverServer : Key → in ClientSend Unit
      SecureReceiverServer(session_key) ≡
        /* Receive some data from client */
        let encrypted_message = ClientSend ? in
          let message = Encrypt(encrypted_message,session_key) in
            /* Process data received */
            SecureReceiverServer(session_key)
          end
        end
    end
```

Appendix G

DocSys – Communication Architecture

G.1 client.rsl

```
comlayer,
clientconnection,
data
scheme Client(D : Data, Com : ComLayer(D)) =
  class
    value
      Client : Nat → in  {Com.L[i].Listen | i : Com.Srv_Rng},
                         {Com.C[i].Client | i : Com.Con_Rng}
                      out {Com.C[i].Server | i : Com.Con_Rng}
                      Unit
      Client(serveraddress) ≡
        let con_no = Com.Connect(serveraddress) in
          ClientConnect(con_no)
        end,

      ClientConnect : Int → in  {Com.C[i].Client | i : Com.Con_Rng}
                             out {Com.C[i].Server | i : Com.Con_Rng}
                             Unit
      ClientConnect(con_no) ≡
        local
          object
            CC : ClientConnection(D,Com.C[con_no])
        in
          CC.ClientConnection()
        end
  end
```

G.2 clientadminlogic.rsl

```
connection,
data
scheme ClientAdminLogic(D : Data, Con : Connection(D)) =
  class
    value
      AdminLogic : Unit → in Con.Client out Con.Server Unit
      AdminLogic() ≡
```

```
        /* Generate and send request*/
        Con.ClientSend(D.mk_admin(D.CreatePerson)) ⌈⌉
        Con.ClientSend(D.mk_admin(D.RemovePerson));
        /* Receive and process response */
        case Con.ClientReceive() of
          D.mk_reply(reply) →
            case reply of
              D.OK → AdminLogic(),
              D.Error → chaos
            end,
          _ → chaos
        end
  end
```

G.3 clientbusinesslogic.rsl

```
connection,
data
scheme ClientBusinessLogic(D : Data, Con : Connection(D)) =
  class
    value
      BusinessLogic : Unit → in Con.Client out Con.Server Unit
      BusinessLogic() ≡
        /* Generate and send request*/
        let placeid,id : Nat in
          Con.ClientSend(D.mk_cmd((placeid,id),D.CreateDoc)) ⌈⌉
          Con.ClientSend(D.mk_cmd((placeid,id),D.PutDocInDir))
        end;
        /* Receive and process response */
        case Con.ClientReceive() of
          D.mk_reply(reply) →
            case reply of
              D.OK → BusinessLogic(),
              D.Error → chaos
            end,
          _ → chaos
        end
  end
```

G.4 clientconnection.rsl

```
connection,
clientforeignlogic,
clientadminlogic,
clientbusinesslogic,
data
scheme ClientConnection(D : Data, Con : Connection(D)) =
  class
    value
      ClientConnection : Unit → in Con.Client out Con.Server Unit
      ClientConnection() ≡
        if ∼Con.ClientAuthenticate() then chaos end;
        Con.ClientProvideIdentification();
        Con.AwaitSecureConnection();
        InstBusinessLogic() ⌈⌉
        InstAdminLogic() ⌈⌉
        InstForeignLogic(),

      InstBusinessLogic : Unit → in Con.Client out Con.Server Unit
      InstBusinessLogic() ≡
```

```
        local
          object
            BCL : ClientBusinessLogic(D,Con)
        in
          Con.ClientSend(D.mk_connectiontype(D.Business));
          BCL.BusinessLogic()
        end,

      InstAdminLogic : Unit → in Con.Client out Con.Server Unit
      InstAdminLogic() ≡
        local
          object
            BCL : ClientAdminLogic(D,Con)
        in
          Con.ClientSend(D.mk_connectiontype(D.Admin));
          BCL.AdminLogic()
        end,

      InstForeignLogic : Unit → in Con.Client out Con.Server Unit
      InstForeignLogic() ≡
        local
          object
            BCL : ClientForeignLogic(D,Con)
        in
          Con.ClientSend(D.mk_connectiontype(D.Foreign));
          BCL.ForeignLogic()
        end
  end
```

G.5 clientforeignlogic.rsl

```
connection,
data
scheme ClientForeignLogic(D : Data, Con : Connection(D)) =
  class
    value
      ForeignLogic : Unit → in Con.Client out Con.Server Unit
      ForeignLogic() ≡
        /* Push or pull data */
        Con.ClientSend(D.mk_foreign(D.PullData)) ⌈⌉
        Con.ClientSend(D.mk_foreign(D.PushData));
        /* Receive and process response */
        case Con.ClientReceive() of
          D.mk_reply(reply) →
            case reply of
              D.OK → ForeignLogic(),
              D.Error → chaos
            end,
          _ → chaos
        end
  end
```

G.6 comlayer.rsl

```
connection
scheme ComLayer(D : class type Data end) =
  class
    object
      C[i : Con_Rng] : Connection(D),
      L[i : Srv_Rng] : class channel Listen : Int end
```

```

   type
     Con_Rng = {|n : Nat • 1 ≤ n ∧ n ≤ max_con|},
     Srv_Rng = {|n : Nat • 1 ≤ n ∧ n ≤ max_serv|}

   value
     max_con : Nat,
     max_serv : Nat

   variable
     count : Int := 0

   value
     Accept : Nat → out {L[i].Listen | i : Srv_Rng}
                    read count
                    write count
                    Int
     Accept(address) ≡
       count := count+1;
       L[address].Listen ! count;
       count,

     Connect : Nat → in {L[i].Listen | i : Srv_Rng}
                     Int
     Connect(address) ≡
       let con_no = L[address].Listen? in
         con_no
       end
 end
```

G.7 commands.rsl

```
place,
dblayer,
data
scheme Commands(D : Data, DB : DBLayer(D)) =
  class
    object
      Place : Place(D,DB)

    value
      CreateDoc : D.ID → Unit
      CreateDoc(id) ≡
        Place.CreateDoc(id),

      PutDocInDir : D.ID → Unit
      PutDocInDir(id) ≡
        Place.PutDocInDir(id)
  end
```

G.8 connection.rsl

```
scheme Connection(D : class type Data end) =
    class
      channel
        Client : D.Data,
        Server : D.Data

      value
        ClientSend : D.Data → out Server Unit
```

```
          ClientSend(t) ≡
            Server ! t,

          ClientReceive : Unit → in Client D.Data
          ClientReceive() ≡
            let t = Client ? in
              t
            end,

          ServerSend : D.Data → out Client Unit
          ServerSend(t) ≡
            Client ! t,

          ServerReceive : Unit → in Server D.Data
          ServerReceive() ≡
            let t = Server ? in
              t
            end

        value
          ServerAuthenticate : Unit → Bool,
          ServerProvideIdentification : Unit → Unit,
          ClientAuthenticate : Unit → Bool,
          ClientProvideIdentification : Unit → Unit,
          EstablishSecureConnection : Unit → Unit,
          AwaitSecureConnection : Unit → Unit
      end
```

G.9 data.rsl

```
scheme Data =
  class
    type
      Data == mk_cmd(ID × Command) |
              mk_admin(AdminCommand) |
              mk_foreign(ForeignCommand) |
              mk_reply(Reply) |
              mk_connectiontype(ConnectionType)

    type
      ID = PlaceID × Nat,
      PlaceID = Nat

    type
      Command == CreateDoc | PutDocInDir,
      AdminCommand == CreatePerson | RemovePerson,
      ForeignCommand == PullData | PushData

    type
      Reply == OK | Error,
      ConnectionType == Business | Admin | Mirror | Foreign

  end
```

G.10 dblayer.rsl

```
data
scheme DBLayer(D : Data) =
  class
    value
```

```
    CreateDoc : D.ID → Unit,
    PutDocInDir : D.ID → Unit
  end
```

G.11 place.rsl

```
dblayer,
data
scheme Place(D : Data, DB : DBLayer(D)) =
  class
    value
      CreateDoc : D.ID → Unit
      CreateDoc(id) ≡
        DB.CreateDoc(id),

      PutDocInDir : D.ID → Unit
      PutDocInDir(id) ≡
        DB.PutDocInDir(id)
  end
```

G.12 server.rsl

```
dblayer,
mdblayer,
comlayer,
serverconnection,
data
scheme Server(D : Data, Com : ComLayer(D)) =
  class
    value
      Server : Nat → in {Com.C[i].Server | i : Com.Con_Rng}
                     out {Com.L[i].Listen | i : Com.Con_Rng},
                         {Com.C[i].Client | i : Com.Con_Rng}
                     read Com.count
                     write Com.count
                     Unit
      Server(serveraddress) ≡
        let con_no = Com.Accept(serveraddress) in
            ServerConnect(con_no) ∥ Server(serveraddress)
        end,

      Server : Nat × Nat → in {Com.C[i].Server | i : Com.Con_Rng},
                              {Com.C[i].Client | i : Com.Con_Rng},
                              {Com.L[i].Listen | i : Com.Con_Rng}
                           out {Com.L[i].Listen | i : Com.Con_Rng},
                               {Com.C[i].Server | i : Com.Con_Rng},
                               {Com.C[i].Client | i : Com.Con_Rng}
                     read Com.count
                     write Com.count
                     Unit
      Server(serveraddress, mirroraddress) ≡
        let con_no = Com.Accept(serveraddress) in
          let mcon_no = Com.Connect(mirroraddress) in
            if ∼Com.C[mcon_no].ClientAuthenticate() then chaos end;
            Com.C[mcon_no].ClientProvideIdentification();
            Com.C[mcon_no].AwaitSecureConnection();
            Com.C[mcon_no].ClientSend(D.mk_connectiontype(D.Mirror));
            MirrorConnect(con_no,mcon_no) ∥
            Server(serveraddress, mirroraddress)
          end
```

```
      end,

    ServerConnect : Int → in {Com.C[i].Server | i : Com.Con_Rng}
                     out {Com.C[i].Client | i : Com.Con_Rng}
                     Unit
    ServerConnect(con_no) ≡
      local
        object
          DB : DBLayer(D),
          SC : ServerConnection(D,Com.C[con_no],DB)
      in
        SC.ServerConnection()
      end,

    MirrorConnect : Int × Int → in {Com.C[i].Server | i : Com.Con_Rng}
                           out {Com.C[i].Client | i : Com.Con_Rng}
                           Unit
    MirrorConnect(con_no,mcon_no) ≡
      local
        object
          MDB : MDBLayer(D,Com.C[mcon_no]),
          SC : ServerConnection(D,Com.C[con_no], MDB)
      in
        SC.ServerConnection()
      end
  end
```

G.13 serveradminlogic.rsl

```
connection,
commands,
dblayer,
data
scheme ServerAdminLogic(D : Data, Con : Connection(D), DB : DBLayer(D)) =
  class
    object
      Commands : Commands(D,DB)

    value
      AdminLogic : Unit → out Con.Client in Con.Server Unit
      AdminLogic() ≡
        case Con.ServerReceive() of
          D.mk_admin(cmd) →
            case cmd of
              D.CreatePerson → /* Add new person to place */
                               Con.ServerSend(D.mk_reply(D.OK)),
              D.RemovePerson → /* Remove person from place */
                               Con.ServerSend(D.mk_reply(D.OK)),
              _ → chaos
            end,
          _ → chaos
        end;
        AdminLogic()
  end
```

G.14 serverbusinesslogic.rsl

```
connection,
commands,
dblayer,
```

```
data
scheme ServerBusinessLogic(D : Data, Con : Connection(D), DB : DBLayer(D)) =
  class
    object
      Commands : Commands(D,DB)

    value
      BusinessLogic : Unit → out Con.Client in Con.Server Unit
      BusinessLogic() ≡
        case Con.ServerReceive() of
          D.mk_cmd(id,cmd) →
            case cmd of
              D.CreateDoc → Commands.CreateDoc(id);
                            Con.ServerSend(D.mk_reply(D.OK)),
              D.PutDocInDir → Commands.PutDocInDir(id);
                              Con.ServerSend(D.mk_reply(D.OK)),
              _ → chaos
            end,
          _ → chaos
        end;
        BusinessLogic()
  end
```

G.15 serverconnection.rsl

```
connection,
serverforeignlogic,
serveradminlogic,
servermirrorlogic,
serverbusinesslogic,
dblayer,
data
scheme ServerConnection(D : Data, Con : Connection(D), DB : DBLayer(D)) =
  class
    value
      ServerConnection : Unit → in Con.Server out Con.Client Unit
      ServerConnection() ≡
        Con.ServerProvideIdentification();
        if ∼Con.ServerAuthenticate() then chaos end;
        Con.EstablishSecureConnection();
        case Con.ServerReceive() of
          D.mk_connectiontype(con_type) →
            case con_type of
              D.Business → InstBusinessLogic(),
              D.Mirror → InstMirrorLogic(),
              D.Admin → InstAdminLogic(),
              _ → chaos
            end,
          _ → chaos
        end,

      InstBusinessLogic : Unit → in Con.Server out Con.Client Unit
      InstBusinessLogic() ≡
        local
          object
            BSL : ServerBusinessLogic(D,Con,DB)
        in
          BSL.BusinessLogic()
        end,

      InstMirrorLogic : Unit → in Con.Server out Con.Client Unit
      InstMirrorLogic() ≡
```

```
        local
          object
            LDB : DBLayer(D),
            ML  : ServerMirrorLogic(D,Con,LDB)
        in
          ML.MirrorLogic()
        end,

      InstAdminLogic : Unit → in Con.Server out Con.Client Unit
      InstAdminLogic() ≡
        local
          object
            LDB : DBLayer(D),
            SAL : ServerAdminLogic(D,Con,LDB)
        in
          SAL.AdminLogic()
        end,

      InstForeignLogic : Unit → in Con.Server out Con.Client Unit
      InstForeignLogic() ≡
        local
          object
            LDB : DBLayer(D),
            SAL : ServerForeignLogic(D,Con,LDB)
        in
          SAL.ForeignLogic()
        end
  end
```

G.16 serverforeignlogic.rsl

```
connection,
commands,
dblayer,
data
scheme ServerForeignLogic(D : Data, Con : Connection(D), DB : DBLayer(D)) =
  class
    object
      Commands : Commands(D,DB)

    value
      ForeignLogic : Unit → out Con.Client in Con.Server Unit
      ForeignLogic() ≡
        case Con.ServerReceive() of
          D.mk_foreign(cmd) →
            case cmd of
              D.PullData → /* Extract data from DB */
                           Con.ServerSend(D.mk_reply(D.OK)),
              D.PushData → /* Insert data into DB */
                           Con.ServerSend(D.mk_reply(D.OK)),
              _ → chaos
            end,
          _ → chaos
        end;
        ForeignLogic()
  end
```

G.17 servermirrorlogic.rsl

```
connection,
```

```
commands,
dblayer,
data
scheme ServerMirrorLogic(D : Data, Con : Connection(D), DB : DBLayer(D)) =
  class
    object
      Commands : Commands(D,DB)

    value
      MirrorLogic : Unit → out Con.Client in Con.Server Unit
      MirrorLogic() ≡
        case Con.ServerReceive() of
          D.mk_cmd(id,cmd) →
            case cmd of
              D.CreateDoc → Commands.CreateDoc(id);
                             Con.ServerSend(D.mk_reply(D.OK)),
              D.PutDocInDir → Commands.PutDocInDir(id);
                               Con.ServerSend(D.mk_reply(D.OK)),
              _ → chaos
            end,
          _ → chaos
        end;
        MirrorLogic()
  end
```

G.18 system.rsl

```
comlayer,
server,
mirror,
client,
data
scheme System =
  class
    object
      D : Data,
      Com : ComLayer(D),
      S : Server(D,Com),
      M : Mirror(D,Com),
      C[i : Client_Range] : Client(D,Com)

    type
      Client_Range = {|n : Nat • 1 ≤ n ∧ n ≤ client_no|}

    value
      client_no : Nat

    value
      System : Unit →  in  {Com.L[i].Listen | i : Com.Srv_Rng},
                           {Com.C[i].Client | i : Com.Con_Rng},
                           {Com.C[i].Server | i : Com.Con_Rng}
                       out {Com.L[i].Listen | i : Com.Srv_Rng},
                           {Com.C[i].Client | i : Com.Con_Rng},
                           {Com.C[i].Server | i : Com.Con_Rng}
                       read Com.count
                       write Com.count
                       Unit
      System() ≡
        S.Server(1,10)  ∥  M.Mirror(10,{1})  ∥ ∥  {C[i].Client(1) | i : Client_Range}
  end
```

G.19 mdblayer.rsl

```
connection,
dblayer,
data
scheme MDBLayer(D : Data, Con : Connection(D)) =
  extend DBLayer(D) with
  class
    axiom
      ∀ id : D.ID •
        CreateDoc(id) ≡
          Con.ClientSend(D.mk_cmd(id,D.CreateDoc)),

      ∀ id : D.ID •
        PutDocInDir(id) ≡
          Con.ClientSend(D.mk_cmd(id,D.PutDocInDir))
  end
```

G.20 mirror.rsl

```
dblayer,
comlayer,
mirrorconnection
scheme Mirror(D : Data, Com : ComLayer(D)) =
  class
    value
      Mirror : Nat × Nat-set → in {Com.C[i].Server | i : Com.Con_Rng},
                                  {Com.L[i].Listen | i : Com.Srv_Rng},
                                  {Com.C[i].Client | i : Com.Con_Rng}
                               out {Com.L[i].Listen | i : Com.Srv_Rng},
                                   {Com.C[i].Server | i : Com.Con_Rng},
                                   {Com.C[i].Client | i : Com.Con_Rng}
                               read Com.count
                               write Com.count
                               Unit
      Mirror(mirroraddress, servers) ≡
        let con_no = Com.Accept(mirroraddress) in
          MirrorConnect(con_no, servers) ∥ Mirror(mirroraddress,servers)
        end,

      MirrorConnect : Nat × Nat-set → in {Com.C[i].Server | i : Com.Con_Rng},
                                         {Com.L[i].Listen | i : Com.Srv_Rng
      },
                                         {Com.C[i].Client | i : Com.Con_Rng}
                                      out {Com.L[i].Listen | i : Com.Srv_Rng
      },
                                          {Com.C[i].Server | i : Com.Con_Rng
      },
                                          {Com.C[i].Client | i : Com.Con_Rng
      }
                                      Unit
      MirrorConnect(con_no, servers) ≡
        local
          object
            DB : DBLayer(D),
            MC : MirrorConnection(D,Com,Com.C[con_no],DB)
        in
          MC.MirrorConnection(servers)
        end
  end
```

G.21 mirroradminlogic.rsl

```
connection,
commands,
dblayer,
data
scheme MirrorAdminLogic(D : Data, Con : Connection(D), DB : DBLayer(D)) =
  class
    object
      Commands : Commands(D,DB)

    value
      AdminLogic : Unit → out Con.Client in Con.Server Unit
      AdminLogic() ≡
        case Con.ServerReceive() of
          D.mk_admin(cmd) →
            case cmd of
              D.CreatePerson → /* Create person in DB */
                                  Con.ServerSend(D.mk_reply(D.OK)),
              D.RemovePerson → /* Remove person from DB */
                                  Con.ServerSend(D.mk_reply(D.OK)),
              _ → chaos
            end,
          _ → chaos
        end;
        AdminLogic()
  end
```

G.22 mirrorconnection.rsl

```
connection,
mirrorforeignlogic,
mirroradminlogic,
mirrorlogic,
dblayer,
data
scheme MirrorConnection(D : Data, Com : ComLayer(D), Con : Connection(D), DB :
     DBLayer(D)) =
  class
    value
      MirrorConnection : Nat-set → in Con.Server,
                                      {Com.L[i].Listen | i : Com.Srv_Rng},
                                      {Com.C[i].Client | i : Com.Con_Rng}
                                   out Con.Client,
                                       {Com.C[i].Server | i : Com.Con_Rng}
                                   Unit

      MirrorConnection(servers) ≡
        Con.ServerProvideIdentification();
        if ~Con.ServerAuthenticate() then chaos end;
        Con.EstablishSecureConnection();
        case Con.ServerReceive() of
          D.mk_connectiontype(con_type) →
            case con_type of
              D.Mirror → InstMirrorLogic(servers),
              D.Admin → InstAdminLogic(),
              _ → chaos
            end,
          _ → chaos
        end,

```

```
      InstMirrorLogic : Nat-set → in Con.Server,
                                       {Com.L[i].Listen | i : Com.Srv_Rng},
                                       {Com.C[i].Client | i : Com.Con_Rng}
                                    out Con.Client,
                                        {Com.C[i].Server | i : Com.Con_Rng}
                                    Unit
      InstMirrorLogic(servers) ≡
        local
          object
            ML : MirrorLogic(D,Com,Con,DB)
        in
          ML.MirrorLogic(servers)
        end,

      InstAdminLogic : Unit → in Con.Server out Con.Client Unit
      InstAdminLogic() ≡
        local
          object
            MAL : MirrorAdminLogic(D,Con,DB)
        in
          MAL.AdminLogic()
        end,

      InstForeignLogic : Unit → in Con.Server out Con.Client Unit
      InstForeignLogic() ≡
        local
          object
            MFL : MirrorForeignLogic(D,Con,DB)
        in
          MFL.ForeignLogic()
        end
  end
```

G.23 mirrorforeignlogic.rsl

```
connection,
commands,
dblayer,
data
scheme MirrorForeignLogic(D : Data, Con : Connection(D), DB : DBLayer(D)) =
  class
    object
      Commands : Commands(D,DB)

    value
      ForeignLogic : Unit → out Con.Client in Con.Server Unit
      ForeignLogic() ≡
        case Con.ServerReceive() of
          D.mk_foreign(cmd) →
            case cmd of
              D.PullData → /* Extract data from DB */
                              Con.ServerSend(D.mk_reply(D.OK)),
              D.PushData → /* Insert data into DB */
                              Con.ServerSend(D.mk_reply(D.OK)),
              _ → chaos
            end,
          _ → chaos
        end;
        ForeignLogic()
  end
```

G.24 mirrorlogic.rsl

```
connection,
commands,
dblayer,
comlayer
scheme MirrorLogic(D : Data, Com : ComLayer(D), Con : Connection(D), DB : DBLayer(D)) =
  class
    object
      Commands : Commands(D,DB)

    value
      MirrorLogic : Nat-set → out Con.Client,
                                    {Com.C[i].Server | i : Com.Con_Rng}
                                in Con.Server,
                                   {Com.L[i].Listen | i : Com.Srv_Rng},
                                   {Com.C[i].Client | i : Com.Con_Rng}
                                Unit
      MirrorLogic(servers) ≡
        case Con.ServerReceive() of
          D.mk_cmd((placeid,id),cmd) →
            if placeid ∉ servers then chaos end;
            let con_no = Com.Connect(placeid) in
              if ∼Com.C[con_no].ClientAuthenticate() then chaos end;
              Com.C[con_no].ClientProvideIdentification();
              Com.C[con_no].AwaitSecureConnection();
              Com.C[con_no].ClientSend(D.mk_connectiontype(D.Mirror));
              case cmd of
                D.CreateDoc   → Com.C[con_no].ClientSend(D.mk_cmd((placeid,id), D.CreateDoc));
                                Commands.CreateDoc(placeid,id);
                                Con.ServerSend(D.mk_reply(D.OK)),
                D.PutDocInDir → Com.C[con_no].ClientSend(D.mk_cmd((placeid,id), D.PutDocInDir));
                                Commands.PutDocInDir(placeid,id);
                                Con.ServerSend(D.mk_reply(D.OK)),
                _ → chaos
              end
            end,
          _ → chaos
        end;
        MirrorLogic(servers)
  end
```

Appendix H

DocSys – Implementation

H.1 DSCommands.h

```
// DSCommands.h: interface for the DSCommands class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_DSCOMMANDS_H__A6B32622_E1C5_4941_93CD_D6BA5BA17830__INCLUDED_)
#define AFX_DSCOMMANDS_H__A6B32622_E1C5_4941_93CD_D6BA5BA17830__INCLUDED_

#include "DSDocument.h" // Added by ClassView
#include "afx.h"
#include "DSPerson.h"
#include "DSPlaceID.h"
#include "DSTime.h"
#include "DSContents.h"
#include "DSPlace.h"
#include "../DBLayer/DSDBLayer.h"
#include "DSLocationID.h"
#include "DSExportID.h"
#include "DSDossier.h"
#include "DSSet.h"
#include "DSError.h"

#define DSCreateDoc          "1"
#define DSCreateDos          "2"
#define DSCopy          "3"
#define DSEdit          "4"
#define DSRemoveDoc          "5"
#define DSRemoveDos          "23"
#define DSGetDocFromDos      "6"
#define DSPutDocInDos        "7"
#define DSGetDosFromDos      "8"
#define DSPutDosInDos        "9"
#define DSGetDocFromDir      "10"
#define DSPutDocInDir        "11"
#define DSGetDosFromDir      "12"
#define DSPutDosInDir        "13"
#define DSExport        "14"
#define DSSignDocument       "15"
#define DSResetDocMembership  "16"
#define DSResetDosMembership  "17"
#define DSSendDoc       "18"
#define DSSendDos       "19"
```

```
#define DSSetDocPermission    "20"
#define DSSetDosPermission    "21"
#define DSMerge               "22"
#define DSReturnDoc           "24"
#define DSReturnDos           "25"
#define DSReadDocument        "26"


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class DSCommands
{
public:
    void ReturnDos(DSPersonID& perid, DSTime& time, DSDossierID& dosid);
    void ReturnDoc(DSPersonID& perid, DSTime& time, DSDocumentID& docid);
    void RemoveDos(DSPersonID& perid, DSTime& time, DSDossierID& dosid);
    DSDocumentID CreateDoc(DSPersonID& perid, DSTime& time, CString strDesc,
      DSContents& cont);
    DSDossierID CreateDos(DSPersonID& perid, DSTime& time, DSDossierDescription
       &desc);
    DSDocumentID Copy(DSPersonID& perid, DSTime& time, DSDocumentID& docid);
    DSDocumentID Edit(DSPersonID& perid, DSTime& time, DSDocumentID& docid,
      DSContents& cont);
    void RemoveDoc(DSPersonID& perid, DSTime& time, DSDocumentID& docid);
    void GetDocFromDos(DSPersonID& perid, DSTime& time, DSDossierID& dosid,
      DSDocumentID& docid);
    void PutDocInDos(DSPersonID& perid, DSTime& time, DSDossierID& dosid,
      DSDocumentID& docid);
    void GetDosFromDos(DSPersonID& perid, DSTime& time, DSDossierID& outer_id,
      DSDossierID& inner_id);
    void PutDosInDos(DSPersonID& perid, DSTime& time, DSDossierID& outer_id,
      DSDossierID& inner_id);
    void GetDocFromDir(DSPersonID& perid, DSTime& time, DSIndexID& idxid,
      DSDocumentID& docid);
    void PutDocInDir(DSPersonID& perid, DSTime& time, DSIndexID& idxid,
      DSDocumentID& docid);
    void GetDosFromDir(DSPersonID& perid, DSTime& time, DSIndexID& idxid,
      DSDossierID& dosid);
    void PutDosInDir(DSPersonID& perid, DSTime& time, DSIndexID& idxid,
      DSDossierID& dosid);
    void Export(DSPersonID& perid, DSTime &time, DSLocationID& locid,
      DSDocumentID& docid);
    void SignDocument(DSPersonID& perid, DSTime& time, DSDocumentID& docid,
      DSSignature& sig);
    DSDocumentID Merge(DSPersonID &perid, DSTime& time, DSDocumentID &docid);
    void SendDos(DSPersonID &perid, DSTime &time, DSPersonID &dest_perid,
      DSDossierID &dosid);
    void SendDoc(DSPersonID &perid, DSTime &time, DSPersonID &dest_perid,
      DSDocumentID &docid);
    void SetDosPermission(DSPersonID &perid, DSTime &time, DSDossierID &dosid,
      DSSet &keys, CString &strCmd);
    void SetDocPermission(DSPersonID &perid, DSTime &time, DSDocumentID &docid,
      DSSet &keys, CString &strCmd);
    void ResetDosMembership(DSPersonID &perid, DSTime &time, DSDossierID &dosid)
      ;
    void ResetDocMembership(DSPersonID &perid, DSTime &time, DSDocumentID &docid
      );
    DSCommands(DSDBLayer* db);
    virtual ~DSCommands();

```

```
public:
   DSDocument ReadDocument(DSPersonID& perid, DSTime& time, DSDocumentID& docid
     );


   DSPlace* Place();
   DSDBLayer* m_pDatabase;

private:

   DSPlace* m_pPlace;
};

#endif // !defined(
     AFX_DSCOMMANDS_H__A6B32622_E1C5_4941_93CD_D6BA5BA17830__INCLUDED_)
```

H.2 contents_example.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 2 U (http://www.xmlspy.com) by DiabloDiab (DiabloDiab) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="contents">
    <xs:complexType>
      <xs:choice>
        <xs:annotation>
          <xs:documentation>Choice</xs:documentation>
        </xs:annotation>
        <xs:element name="personinfo">
          <xs:complexType>
            <xs:choice>
              <xs:annotation>
                <xs:documentation>Choice</xs:documentation>
              </xs:annotation>
              <xs:element name="version_01">
                <xs:annotation>
                  <xs:documentation>@tablename = "table1"</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:sequence>
                    <xs:annotation>
                      <xs:documentation>sequence</xs:documentation>
                    </xs:annotation>
                    <xs:element name="name">
                      <xs:annotation>
                        <xs:documentation>value = "John Doe"</xs:documentation>
                      </xs:annotation>
                    </xs:element>
                    <xs:element name="address">
                      <xs:annotation>
                        <xs:documentation>value = "Oak Street 1"</xs:documentation>
                      </xs:annotation>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="tablename" type="xs:string" use="required" fixed="table1"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="version_02">
                <xs:annotation>
                  <xs:documentation>@tablename = "table1"</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:sequence>
                    <xs:annotation>
                      <xs:documentation>sequence</xs:documentation>
                    </xs:annotation>
                    <xs:element name="name">
                      <xs:annotation>
                        <xs:documentation>@tablename = "table2"</xs:documentation>
                      </xs:annotation>
                      <xs:complexType>
                        <xs:sequence>
                          <xs:annotation>
                            <xs:documentation>sequence</xs:documentation>
```

```
                          </xs:annotation>
                          <xs:element name="firstname">
                            <xs:annotation>
                              <xs:documentation>value = "John"</xs:documentation>
                            </xs:annotation>
                          </xs:element>
                          <xs:element name="lastname">
                            <xs:annotation>
                              <xs:documentation>value = "Doe"</xs:documentation>
                            </xs:annotation>
                          </xs:element>
                        </xs:sequence>
                        <xs:attribute name="tablename" type="xs:string" use="required" fixed="table2"/>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="address">
                      <xs:annotation>
                        <xs:documentation>value = "Oak␣Street␣1"</xs:documentation>
                      </xs:annotation>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="tablename" type="xs:string" use="required" fixed="table1"/>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
        <xs:element name="invoice">
          <xs:complexType>
            <xs:sequence/>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="personinfo">
    <xs:complexType>
      <xs:choice>
        <xs:annotation>
          <xs:documentation>Choice</xs:documentation>
        </xs:annotation>
        <xs:element name="version_01">
          <xs:annotation>
            <xs:documentation>@tablename = "table1"</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:annotation>
                <xs:documentation>sequence</xs:documentation>
              </xs:annotation>
              <xs:element name="name">
                <xs:annotation>
                  <xs:documentation>value = "John␣Doe"</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="address">
                <xs:annotation>
```

```
                  <xs:documentation>value = "Oak␣Street␣1"</xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="tablename" type="xs:string" use="required"
     fixed="table1"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="version_02">
          <xs:annotation>
            <xs:documentation>@tablename = "table1"</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:annotation>
                <xs:documentation>sequence</xs:documentation>
              </xs:annotation>
              <xs:element name="name">
                <xs:annotation>
                  <xs:documentation>@tablename = "table2"</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:sequence>
                    <xs:annotation>
                      <xs:documentation>sequence</xs:documentation>
                    </xs:annotation>
                    <xs:element name="firstname">
                      <xs:annotation>
                        <xs:documentation>value = "John"</xs:documentation>
                      </xs:annotation>
                    </xs:element>
                    <xs:element name="lastname">
                      <xs:annotation>
                        <xs:documentation>value = "Doe"</xs:documentation>
                      </xs:annotation>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="tablename" type="xs:string" use="required
     " fixed="table2"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="address">
                <xs:annotation>
                  <xs:documentation>value = "Oak␣Street␣1"</xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="tablename" type="xs:string" use="required"
     fixed="table1"/>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Appendix I

DocSys – Specification Relationship Example

In order to show the the relationship between specifications and implementation – from domain to requirement to implementation – the following is extracted from the report and presented together.

I.1 Domain Specification

```
M: Command → System → System
M(cmd)(places, docids, dosids) ≡
  case cmd of
    mk_Edit(person, plid, time, document, (te,fe)) →
      let (dir,pers,locs,keys) = places(plid) in
       assert(person ∈ rng pers ∧
           document ∈ obs_Documents(person));
        let doc:Document •
          obs_ID(doc) = obs_ID(document) ∧
          obs_Time(doc) = time ∧
          obs_Contents(doc) = te(obs_Contents(document)) ∧
          obs_Type(doc) = version ∧
          obs_Creator(doc) = obs_ID(person) ∧
          obs_PlaceID(doc) = plid ∧
          obs_Signatures(doc) = {} ∧
          obs_DirMembership(doc) = obs_DirMembership(document) ∧
          obs_PlaceMembership(doc) = obs_PlaceMembership(document) ∧
          obs_Ancestor(doc) = obs_Ancestor(document)
        in
          (places † [plid ↦
            (dir, pers † [obs_ID(person) ↦
              (person \ {document}) ∪ {doc}],
           locs, keys)], docids, dosids)
        end
      end
  end
```

I.2 Requirements Specification

```
M: Command → System → System
M(cmd)(places, docids, dosids, copyids) ≡
  case cmd of
    mk_Edit(person, plid, time, document, (te,fe)) →
      let (dir,pers,locs,bin,keys) = places(plid) in
        let docs = obs_Group(document,docids) in
          assert(hasPermission(person,document,Edit) ∧
          person ∈ rng pers ∧
          mostRecentVersion(document,docs) ∧
          docs ⊂ obs_Documents(person));
            let docid:DocumentID • docid ∉ docids in
              let doc:Document •
                obs_ID(doc) = docid ∧
                obs_Time(doc) = time ∧
                obs_Contents(doc) = te(obs_Contents(document)) ∧
                obs_Type(doc) = version ∧
                obs_Creator(doc) = obs_ID(person) ∧
                obs_PlaceID(doc) = plid ∧
                obs_Ancestor(doc) = mk_did(obs_ID(document)) ∧
                obs_Signatures(doc) = {} ∧
                obs_DirMembership(doc) = obs_DirMembership(document) ∧
                obs_DossierMembership(doc) = obs_DossierMembership(document) ∧
                obs_CommandLocks(doc) = obs_CommandLocks(document) ∧
                obs_Events(doc) = obs_Events(document)
              in
                let evt:Event •
                  evt_type(evt) = Edit ∧
                  evt_executedby(evt) = obs_ID(person) ∧
                  evt_time(evt) = time ∧
                  evt_place(evt) = plid
                in
                  (places † [plid ↦
                    (dir, pers † [obs_ID(person) ↦
                      ((person \ {document}) ∪
                                            {addEvent(document,evt)}) ∪
                            {addEvent(doc,evt)}],
                   locs,bin,keys)], docids ∪ {docid}, dosids, copyids)
  end end end end end,
```

I.3 Implementation Specification

```
DSDocumentID DSCommands::Edit(DSPersonID& perid, DSTime& time,
                              DSDocumentID& docid, DSContents& cont)
{
  DSEvent event;
  DSPerson per = DSPerson(m_pDatabase,perid);
  DSDocument doc = DSDocument(m_pDatabase, docid);
  DSDocument edt = DSDocument(m_pDatabase);
  Assert(per.Contains(docid),ERR_PER_DOES_NOT_CONTAIN_DOC);
  Assert(per.Contains(doc.GetKeys(DSEdit)),ERR_PER_DOES_NOT_CONTAIN_CMD_KEY);
  Assert((doc.m_strNewestEditionId == docid.GetEditionID()),ERR_CANNOT_EDIT_OLD_VERSION);
  Assert((doc.m_strContentsType == cont.m_strContentsType &&
          doc.m_strContentsVersion == cont.m_strContentsVersion),
            ERR_CONT_TYPE_OR_VERSION_MISMATCH);
  edt.m_id = doc.NextEditionID();
  edt.m_creator = perid;
  edt.m_type = doc.m_type;
  edt.m_time = time;
  edt.m_ancestor = doc.m_ancestor;
```

```
    edt.m_strContentsType = doc.m_strContentsType;
    edt.m_strContentsVersion = doc.m_strContentsVersion;
    edt.m_strDesc = doc.m_strDesc;
    edt.m_membership = doc.m_membership;
    edt.Flush();
    edt.SetContents(cont);
    event.m_executedBy = perid;
    event.m_time = time;
    event.m_id = edt.m_id;
    event.strCmd = DSEdit;
    doc.Add(event);
    return edt.m_id;
  }
```

Appendix J

EMR – Template Specification

In order to provide the reader with an idea of where to start when defining templates are here an example. It is started off by explicitly defining types needed in the later template specification, such as fonts, colors and information data types. It is followed by a figure of the note and its specification.

J.1 mrcontents.rsl

```
scheme mrcontents =
  class
    type
      text = section*,
      section = heading × paragraph*,
      heading = format × Char*,
      paragraph = format × Char*

    type
      binary == image | audio,
      image == xray | ct | mr | ekg | eeg,
      audio == dictaphone_recording

    type
      ref = Nat,
      dimension = Real × Real,
      position = Real × Real,
      color == red | pink | black | white | blue,
      width = Real,
      font == arial | times_new_roman,
      size = Int,
      style == normal | italics | bold | underline

    type
      layout = position × dimension × border_layout × color,
      border_layout = color × width,
      format = font × size × style × color

    type
      txt_label = layout × text,
      bin_label = layout × binary,
```

```
      txt_input = layout × ref,
      bin_input = layout × ref,

      contents = group × data,
      data = (ref -m-> text) × (ref -m-> binary),

      group == mk_grp(layout × label* × input* × group*),
      label == mk_ltxt(txt_label) | mk_lbin(bin_label),
      input == mk_itxt(txt_input) | mk_ibin(bin_input)
  end
```

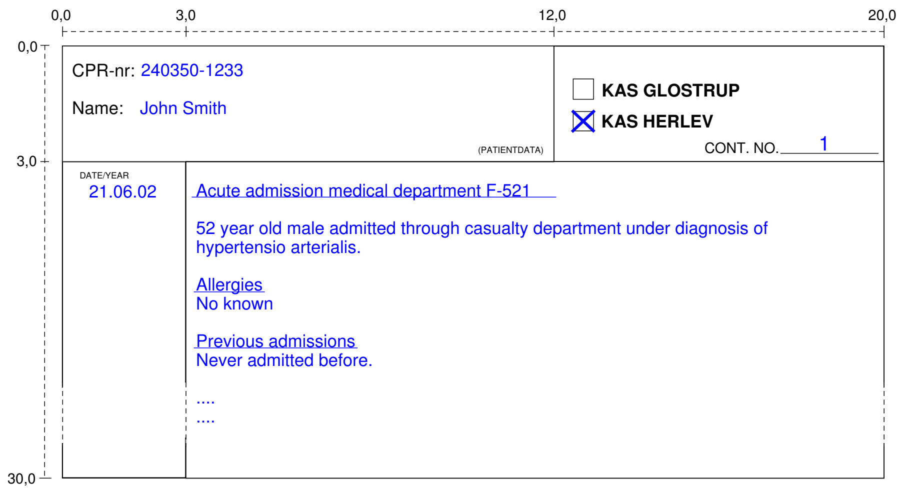


Figure J.1: Note Page of a Medical Record With Data

J.2 mrnote.rsl

```
mrcontents
scheme mrnote =
  extend mrcontents with
  class
    value
      pdformat1 : format = (times_new_roman,6,normal,black),
      pdhead1 : heading = (pdformat1,"(PATIENTDATA)"),
      pdtext1 : text = ⟨(pdhead1,⟨⟩)⟩,
      pdlabel1 : label = mk_ltxt(((9.0,2.5),(3.0,0.2),(white,0.0),white),
     pdtext1),

      pdformat2 : format = (times_new_roman,12,normal,black),
      pdhead2 : heading = (pdformat2,"CPR:"),
      pdtext2 : text = ⟨(pdhead2,⟨⟩)⟩,
      pdlabel2 : label = mk_ltxt(((0.5,0.5),(1.5,0.5),(white,0.0),white),
     pdtext2),
      pdinput2 : input = mk_itxt(((2.0,0.5),(3.0,0.5),(white,0.0),white), 1),

      pdhead3 : heading = (pdformat2,"Name:"),
      pdtext3 : text = ⟨(pdhead3,⟨⟩)⟩,
```

```
    pdlabel3 : label = mk_ltxt(((0.5,1.5),(1.5,0.5),(white,0.0),white),
   pdtext3),
    pdinput3 : input = mk_itxt(((2.0,1.5),(3.5,0.5),(white,0.0),white), 2),

    pdgroup : group = mk_grp(((0.0,0.0),(12.0,3.0),(black,0.2),white),
                             ⟨pdlabel1,pdlabel2,pdlabel3⟩,
                             ⟨pdinput2,pdinput3⟩,
                             ⟨⟩)

  value
    pnformat1 : format = (times_new_roman,12,bold,black),
    pnhead1 : heading = (pnformat1,"CONT. NO.__________"),
    pntext1 : text = ⟨(pnhead1,⟨⟩)⟩,
    pnlabel1 : label = mk_ltxt(((4.0,2.5),(4.0,0.5),(white,0.0),white),
   pntext1),
    pninput1 : input = mk_itxt(((6.0,2.0),(1.0,1.0),(white,0.0),white), 3),

    pnformat2 : format = (times_new_roman,10,normal,black),
    pnhead2 : heading = (pnformat2,"KAS GLOSTRUP"),
    pntext2 : text = ⟨(pnhead2,⟨⟩)⟩,
    pnlabel2 : label = mk_ltxt(((1.0,1.0),(4.0,0.5),(white,0.0),white),
   pntext2),
    pninput2 : input = mk_itxt(((0.5,1.0),(0.5,0.5),(black,0.2),white), 4),

    pnhead3 : heading = (pnformat2,"KAS HERLEV"),
    pntext3 : text = ⟨(pnhead3,⟨⟩)⟩,
    pnlabel3 : label = mk_ltxt(((1.0,1.5),(4.0,0.5),(white,0.0),white),
   pntext3),
    pninput3 : input = mk_itxt(((0.5,1.5),(0.5,0.5),(black,0.2),white), 5),

    pngroup : group = mk_grp(((12.0,0.0),(8.0,3.0),(black,0.2),white),
                             ⟨pnlabel1,pnlabel2,pnlabel3⟩,
                             ⟨pninput1,pninput2,pninput3⟩,
                             ⟨⟩)

  value
    dformat : format = (times_new_roman,6,normal,black),
    dhead : heading = (dformat,"DATE/YEAR"),
    dtext : text = ⟨(dhead,⟨⟩)⟩,
    dlabel : label = mk_ltxt(((0.5,0.1),(1.0,0.2),(white,0.0),white),dtext),
    dinput : input = mk_itxt(((0.5,0.5),(2.0,26.0),(white,0.0),white), 6),

    dgroup : group = mk_grp(((0.0,3.0),(3.0,27.0),(white,0.0),white),
                            ⟨dlabel⟩,
                            ⟨dinput⟩,
                            ⟨⟩)

  value
    tinput : input = mk_itxt(((0.0,0.0),(17.0,27.0),(white,0.0),white), 7),

    tgroup : group = mk_grp(((3.0,3.0),(17.0,27.0),(white,0.0),white),
                            ⟨⟩,
                            ⟨tinput⟩,
                            ⟨⟩)

  variable
    template : group := mk_grp(((0.0,0.0),(20.0,30.0),(white,0.0),white),
                               ⟨⟩,
                               ⟨⟩,
                               ⟨pdgroup,pngroup,dgroup,tgroup⟩)

  value
```

```
    dataformat1 : format = (times_new_roman,12,normal,blue),
    dataformat2 : format = (times_new_roman,12,underline,blue),

    data1head : heading = (dataformat1,"240350-1233"),
    data1 : text = ⟨(data1head,⟨⟩)⟩,
    data2head : heading = (dataformat1,"John Smith"),
    data2 : text = ⟨(data2head,⟨⟩)⟩,
    data3head : heading = (dataformat1,"1"),
    data3 : text = ⟨(data3head,⟨⟩)⟩,
    data4head : heading = (dataformat1,"X"),
    data4 : text = ⟨(data4head,⟨⟩)⟩,
    data5head : heading = (dataformat1,""),
    data5 : text = ⟨(data5head,⟨⟩)⟩,
    data6head : heading = (dataformat1,"21.06.02"),
    data6 : text = ⟨(data6head,⟨⟩)⟩,
    data7head1 : heading = (dataformat2,"Acute admission medical department F
   -521"),
    data7para1 : paragraph = (dataformat1,"52 year old male admitted through
   casualty department under diagnosis of hypertensio arterialis."),
    data7head2 : heading = (dataformat2,"Allergies"),
    data7para2 : paragraph = (dataformat1,"No known"),
    data7head3 : heading = (dataformat2,"Previous admissions"),
    data7para3 : paragraph = (dataformat1,"Never admittet before."),
    data7 : text = ⟨(data7head1,⟨data7para1⟩),(data7head2,⟨data7para2⟩),(
   data7head3,⟨data7para3⟩)⟩,

    d : data = ([1 ↦ data1, 2 ↦ data2, 3 ↦ data3, 4 ↦ data4, 5 ↦ data5, 6 ↦
   data6, 7 ↦ data7],[])

  value
    f : data → read template contents,
    f_inv : contents → data

  axiom ∀ d : data • f_inv(f(d)) = d

end
```

Appendix K

EMR – GUI Design

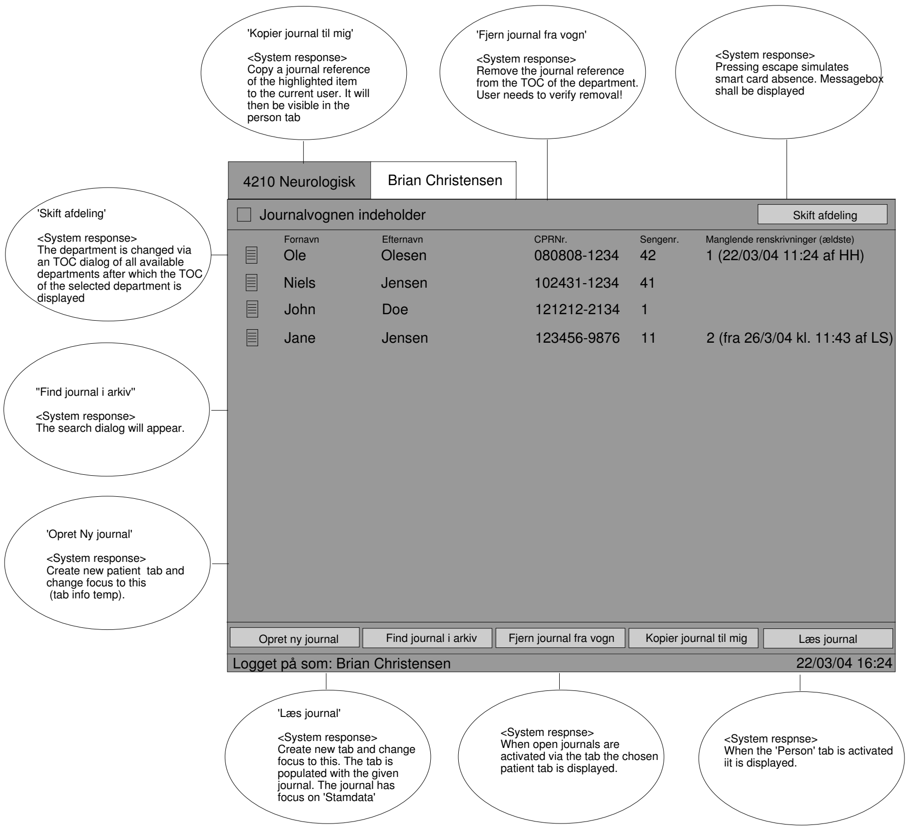


Figure K.1: Department tab layout

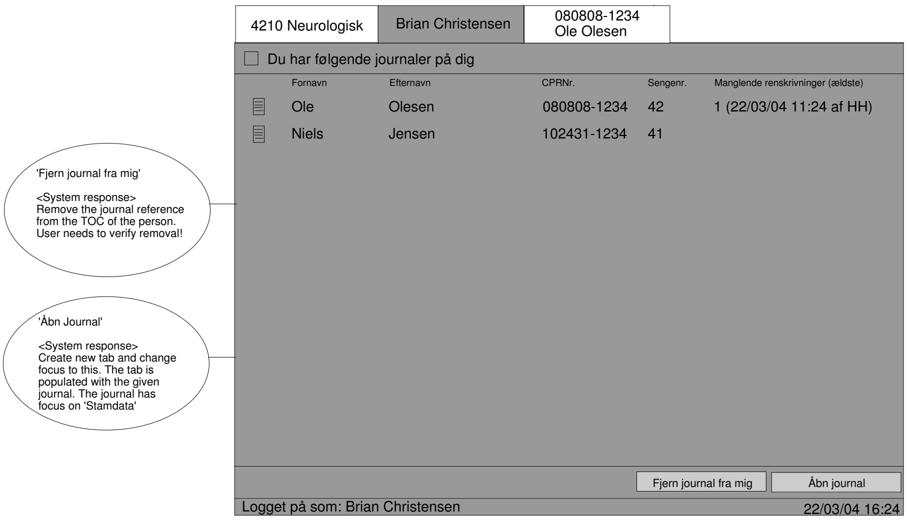


Figure K.2: Person Tab Layout

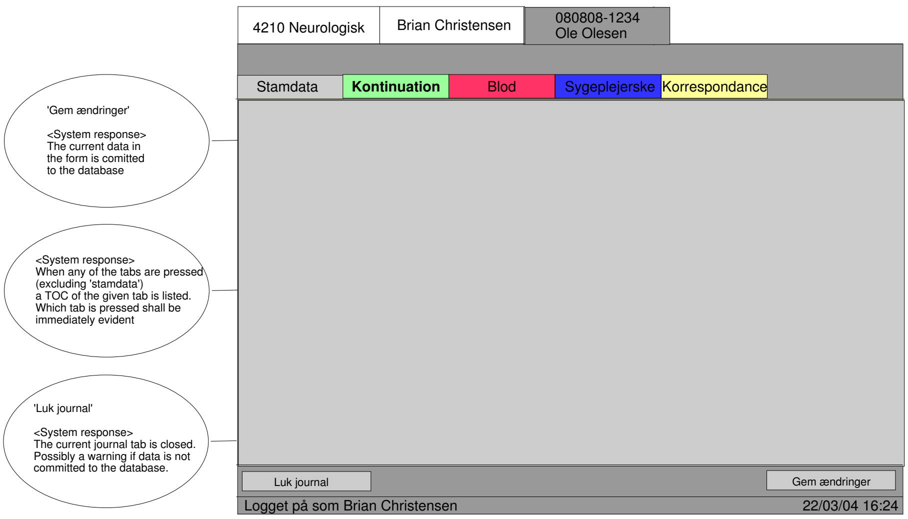


Figure K.3: Patient Tab Layout

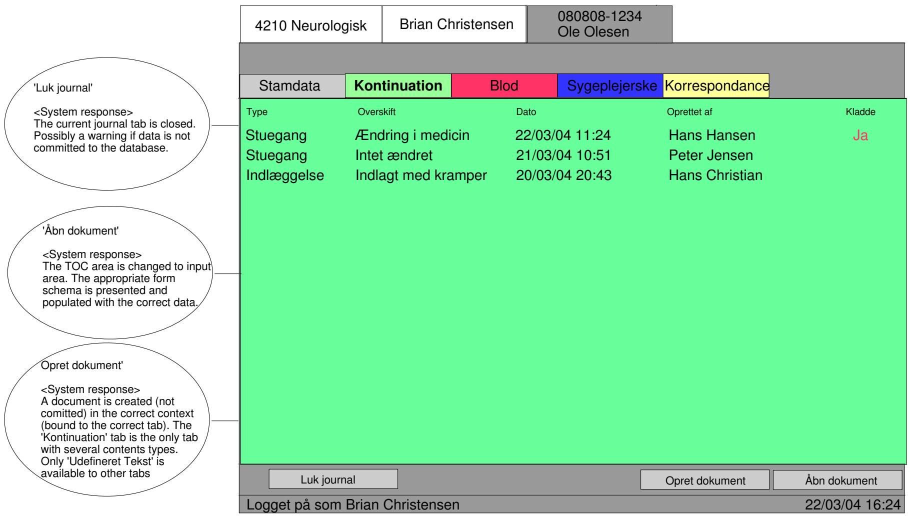


Figure K.4: Medical Record Category Layout

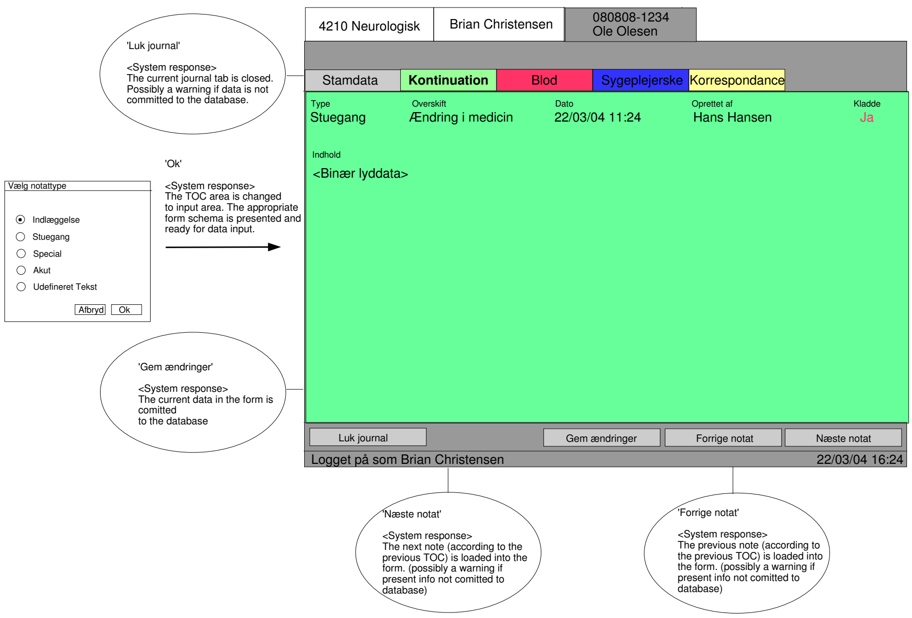


Figure K.5: Medical Record Note Display Layout

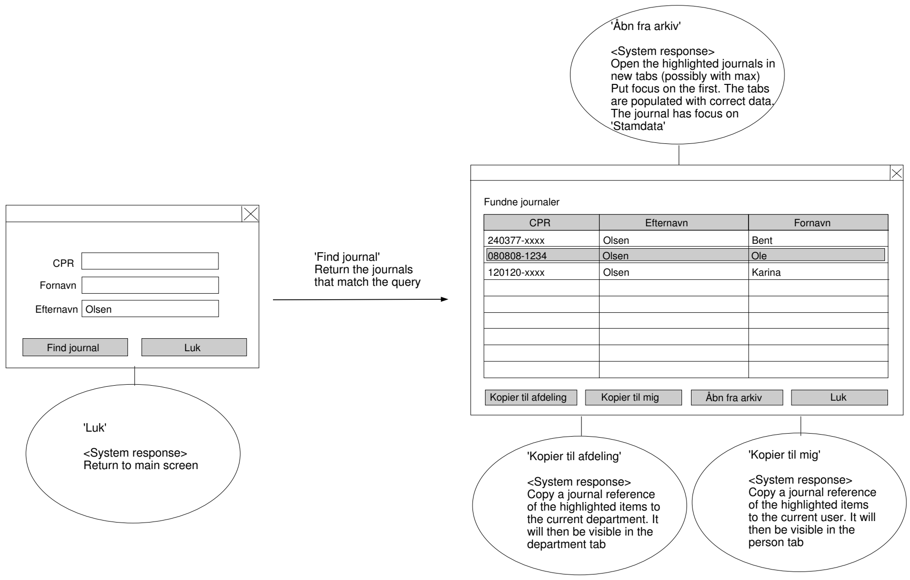


Figure K.6: Search Dialog

Appendix L

EMR – Business Logic

```
void DSBusinessLayerEMR::CommandCreateRecord(DSXMLNode &commandnode)
{
  DSTime time = DSTime(CTime::GetCurrentTime());

  // Parse contents
  DSXMLNode contentsnode = commandnode.FindNode("//contents");
  DSContents contents;
  contents.Get()->LoadSchema("http://www.lindqvist.it/contents.xsd");
  if(!contents.Get()->Parse(&contentsnode.GetText()))
  {
    Error("Error in contents");
    return;
  }

  // set type and version of contents
  contents.m_strContentsType = contents.Get()->FindNode("//contents/*").GetName();
  contents.m_strContentsVersion = contents.Get()->FindNode("//contents/*/*").GetName();
  contents.Get()->FindNode("//_antalkladder").SetText(&CString("0"));

  // create cover data document
  DSDocumentID covdat = m_pCommands->CreateDoc(m_PersonID,time,"_coverdata",contents);
  contents.CleanUp();

  DSDocumentID newref = m_pCommands->Copy(m_PersonID,time,covdat);
  m_pCommands->PutDocInDir(m_PersonID,time,m_DepartmentID,newref);

  // create medical record and place cover data document inside
  DSDossierID emr = m_pCommands->CreateDos(m_PersonID,time,DSDossierDescription("_emr"));
  m_pCommands->PutDocInDos(m_PersonID,time,emr,covdat);

  // create and add remaining subfolders
  m_pCommands->PutDosInDos(m_PersonID,time,emr,m_pCommands->CreateDos(m_PersonID,time,DSDossierDescription("Kontinuation")));
  m_pCommands->PutDosInDos(m_PersonID,time,emr,m_pCommands->CreateDos(m_PersonID,time,DSDossierDescription("Blod")));
  m_pCommands->PutDosInDos(m_PersonID,time,emr,m_pCommands->CreateDos(m_PersonID,time,DSDossierDescription("Sygeplejerske")));
  m_pCommands->PutDosInDos(m_PersonID,time,emr,m_pCommands->CreateDos(m_PersonID,time,DSDossierDescription("Korrespondance")));
```

```

  // Put record in archive
  m_pCommands->PutDosInDir(m_PersonID,time,m_ArchiveID,emr);

  Success(newref);
}
void DSBusinessLayerEMR::CommandCreateNote(DSXMLNode &commandnode)
{
  DSDocument coverdata = DSDocument(m_pDatabase,GetDocumentID(commandnode.
     FindNode("documentid")));

  if(coverdata.m_type.GetType() == DScopy)
    coverdata = DSDocument(m_pDatabase,coverdata.m_ancestor.m_DocumentID);

  DSContents cont;
  cont.Get()->LoadSchema("http://www.lindqvist.it/contents.xsd");
  if(!cont.Get()->Parse(&commandnode.FindNode("contents").GetText()))
  {
    Error("Error␣in␣contents");
    return;
  }
  CString notetype = cont.Get()->FindNode("//contents/*").GetName();
  CString noteversion = cont.Get()->FindNode("//contents/*/*").GetName();
  cont.m_strContentsType = notetype;
  cont.m_strContentsVersion = noteversion;
  DSDocumentID newnote = m_pCommands->CreateDoc(m_PersonID,DSTime(),notetype,
     cont);

  bool getsuccess = false;
  while(!getsuccess)
  {
    try
    {
      m_pCommands->GetDosFromDir(m_PersonID,DSTime(),m_ArchiveID,coverdata.
     m_membership.GetDossierID());
      getsuccess = true;
    }
    catch(DSError e)
    {
      if(e.GetType() != ERR_IDX_DOES_NOT_CONTAIN_DOS) throw e;
      else Sleep(200);
    }
  }

  if(cont.Get()->FindNode("//_kladde").GetText() == "Ja")
  {
    DSDocument newestcoverdata(m_pDatabase,DSDocumentID(coverdata.m_id.
     GetPlaceID(),coverdata.m_id.GetGroupID(),""));
    DSContents covercont = newestcoverdata.GetContents();
    int no_of_drafts = atoi(covercont.Get()->FindNode("//_antalkladder").
     GetText());
    CString str_no_of_drafts;
    str_no_of_drafts.Format("%d",no_of_drafts+1);
    covercont.Get()->FindNode("//_antalkladder").SetText(&str_no_of_drafts);
    m_pCommands->Edit(m_PersonID,DSTime(),newestcoverdata.m_id,covercont);
  }
  cont.CleanUp();

  CString subdossier = commandnode.FindNode("//subdossier").GetText();
  DSDossier dos(m_pDatabase,coverdata.m_membership.GetDossierID());
  DSSet set = dos.GetTOC();
  set.Reset();
```

```
  while(set.HasMore())
  {
    DSObject* elem = set.Next();
    if(elem->IsKindOf(RUNTIME_CLASS(DSDossier)))
    {
      DSDossier* dossier = (DSDossier*)elem;
      if(dossier->m_dossdesc.GetDescription() == subdossier)
      {
        m_pCommands->GetDosFromDos(m_PersonID,DSTime(),dos.m_id,dossier->m_id);
        m_pCommands->PutDocInDos(m_PersonID,DSTime(),dossier->m_id,newnote);
        m_pCommands->ReturnDos(m_PersonID,DSTime(),dossier->m_id);
      }
    }
  }
  set.CleanUp();
  m_pCommands->ReturnDos(m_PersonID,DSTime(),dos.m_id);
  Success(newnote);
}
void DSBusinessLayerEMR::CommandAddToDepartment(DSXMLNode &commandnode)
{
  DSTime time = DSTime(CTime::GetCurrentTime());
  DSDocument coverdata = DSDocument(m_pDatabase,GetDocumentID(commandnode.FindNode("documentid")));

  if(coverdata.m_type.GetType() == DScopy)
    coverdata = DSDocument(m_pDatabase,coverdata.m_ancestor.m_DocumentID);

  DSSet deptoc = m_Department.GetTOC();
  deptoc.Reset();
  while(deptoc.HasMore())
  {
    DSObject* elem = deptoc.Next();
    if(elem->IsKindOf(RUNTIME_CLASS(DSDocument)))
    {
      DSDocument* doc = (DSDocument*)elem;
      if(doc->m_ancestor.m_DocumentID.GetPlaceID().GetID() == coverdata.m_id.GetPlaceID().GetID() && doc->m_ancestor.m_DocumentID.GetGroupID() == coverdata.m_id.GetGroupID())
      {
        deptoc.CleanUp();
        Success();
        return;
      }
    }
  }
  deptoc.CleanUp();

  bool getsuccess = false;
  while(!getsuccess)
  {
    try
    {
      m_pCommands->GetDosFromDir(m_PersonID,time,m_ArchiveID,coverdata.m_membership.GetDossierID());
      getsuccess = true;
    }
    catch(DSError e)
    {
      if(e.GetType() != ERR_IDX_DOES_NOT_CONTAIN_DOS) throw e;
      else Sleep(200);
    }
  }
```

```

  DSDocumentID newref = m_pCommands->Copy(m_PersonID,time,coverdata.m_id);
  m_pCommands->PutDosInDir(m_PersonID,time,m_ArchiveID,coverdata.m_membership.GetDossierID());
  m_pCommands->PutDocInDir(m_PersonID,time,m_DepartmentID,newref);
  Success(newref);
}
void DSBusinessLayerEMR::CommandAddToPerson(DSXMLNode &commandnode)
{
  DSTime time = DSTime(CTime::GetCurrentTime());

  DSDocument coverdata = DSDocument(m_pDatabase,GetDocumentID(commandnode.FindNode("documentid")));

  if(coverdata.m_type.GetType() == DScopy)
    coverdata = DSDocument(m_pDatabase,coverdata.m_ancestor.m_DocumentID);

  DSSet pertoc = m_Person.GetTOC();
  pertoc.Reset();
  while(pertoc.HasMore())
  {
    DSObject* elem = pertoc.Next();
    if(elem->IsKindOf(RUNTIME_CLASS(DSDocument)))
    {
      DSDocument* doc = (DSDocument*)elem;
      if(doc->m_ancestor.m_DocumentID.GetPlaceID().GetID() == coverdata.m_id.GetPlaceID().GetID() && doc->m_ancestor.m_DocumentID.GetGroupID() == coverdata.m_id.GetGroupID())
      {
        pertoc.CleanUp();
        Success();
        return;
      }
    }
  }
  pertoc.CleanUp();

  bool getsuccess = false;
  while(!getsuccess)
  {
    try
    {
      m_pCommands->GetDosFromDir(m_PersonID,time,m_ArchiveID,coverdata.m_membership.GetDossierID());
      getsuccess = true;
    }
    catch(DSError e)
    {
      if(e.GetType() != ERR_IDX_DOES_NOT_CONTAIN_DOS) throw e;
      else Sleep(200);
    }
  }

  DSDocumentID newref = m_pCommands->Copy(m_PersonID,time,coverdata.m_id);
  m_pCommands->PutDosInDir(m_PersonID,time,m_ArchiveID,coverdata.m_membership.GetDossierID());
  Success(newref);
}
void DSBusinessLayerEMR::CommandRemoveFromDepartment(DSXMLNode &commandnode)
{
  DSTime time = DSTime(CTime::GetCurrentTime());
  DSDocumentID docid = GetDocumentID(commandnode.FindNode("documentid"));
```

```

  m_pCommands->GetDocFromDir(m_PersonID,time,m_DepartmentID,docid);
  m_pCommands->RemoveDoc(m_PersonID,time,docid);
  Success();
}
void DSBusinessLayerEMR::CommandRemoveFromPerson(DSXMLNode &commandnode)
{
  DSTime time = DSTime(CTime::GetCurrentTime());
  m_pCommands->RemoveDoc(m_PersonID,time,GetDocumentID(commandnode.FindNode("documentid")));
  Success();
}
void DSBusinessLayerEMR::CommandSaveChanges(DSXMLNode &commandnode)
{
  DSContents contents;
  contents.Get()->LoadSchema("http://www.lindqvist.it/contents.xsd");
  if(!contents.Get()->Parse(&commandnode.FindNode("contents").GetText()))
  {
    Error("Error in contents");
    return;
  }
  CString notetype = contents.Get()->FindNode("//contents/*").GetName();
  CString noteversion = contents.Get()->FindNode("//contents/*/*").GetName();
  contents.m_strContentsType = notetype;
  contents.m_strContentsVersion = noteversion;

  DSDocument note(m_pDatabase,GetDocumentID(commandnode.FindNode("documentid")));
  DSContents contprevedition = note.GetContents();

  DSDossierID rootid = note.m_membership.GetDossierID();
  DSIndexID idxid;

  bool foundroot = false;
  while(!foundroot)
  {
    DSDossier parent(m_pDatabase,rootid);
    idxid = parent.m_membership.GetIndexID();
    if(parent.m_dossdesc.GetDescription()== "_emr") foundroot = true;
    else rootid = parent.m_membership.GetDossierID();
  }

  bool getsuccess = false;
  while(!getsuccess)
  {
    try
    {
      m_pCommands->GetDosFromDir(m_PersonID,DSTime(),idxid,rootid);
      getsuccess = true;
    }
    catch(DSError e)
    {
      if(e.GetType() != ERR_IDX_DOES_NOT_CONTAIN_DOS) throw e;
      else Sleep(200);
    }
  }

  DSDocumentID newref;
  try
  {
       newref = m_pCommands->Edit(m_PersonID,DSTime(),note.m_id,contents);
  }
```

```
  catch(DSError e)
  {
    contprevedition.CleanUp();
    m_pCommands->ReturnDos(m_PersonID,DSTime(),rootid);
    Error("Noten␣er␣blevetæ␣ndret␣af␣en␣anden␣i␣mellemtiden");
    return;

  }

  if(contents.Get()->FindNode("//_kladde").GetText() != contprevedition.Get()->FindNode("//_kladde").GetText())
  {
    int status;
    if(contents.Get()->FindNode("//_kladde").GetText() != "Ja")
      status = -1;
    else
      status = 1;

    DSDossier emrdos(m_pDatabase,rootid);
    DSSet emrtoc = emrdos.GetTOC();
    emrtoc.Reset();
    while(emrtoc.HasMore())
    {
      DSObject* elem = emrtoc.Next();
      if(elem->IsKindOf(RUNTIME_CLASS(DSDocument)))
      {
        DSDocument* doc = (DSDocument*)elem;
        DSDocument newestcover(m_pDatabase,DSDocumentID(doc->m_id.GetPlaceID(),doc->m_id.GetGroupID(),""));
        DSContents covercont = newestcover.GetContents();
        int no_of_drafts = atoi(covercont.Get()->FindNode("//_antalkladder").GetText());
        CString str_no_of_drafts;
        str_no_of_drafts.Format("%d",no_of_drafts+status);
        covercont.Get()->FindNode("//_antalkladder").SetText(&str_no_of_drafts);
        m_pCommands->Edit(m_PersonID,DSTime(),newestcover.m_id,covercont);
        covercont.CleanUp();
      }
    }
    emrtoc.CleanUp();
  }
  m_pCommands->ReturnDos(m_PersonID,DSTime(),rootid);

  contents.CleanUp();
  contprevedition.CleanUp();
  Success(newref);
}
void DSBusinessLayerEMR::CommandChangeDepartment(DSXMLNode &commandnode)
{
  DSTime time = DSTime(CTime::GetCurrentTime());

  DSIndexID newdepid = GetIndexID(commandnode.FindNode("indexid"));
  if(!m_pPlace->Dir()->Contains(newdepid))
  {
    Error("Department␣does␣not␣exist");
    return;
  }

  DSXMLNode dep = m_contPersonPref.Get()->FindNode("//department");
  dep.FindNode("_placeid").SetText(&newdepid.GetPlaceID().GetID());
  dep.FindNode("_localid").SetText(&newdepid.GetID());
```

```
  m_pCommands->Edit(m_PersonID,time,m_docidPersonPref,m_contPersonPref);

  m_bPrefsLoaded = false;

  Success();
}
void DSBusinessLayerEMR::RequestCenterTOC(DSXMLNode &parent)
{
  DSIndexID indexid = DSIndexID(m_pPlace->GetID(),EMR_Centers);
  if(!m_pPlace->Dir()->Contains(indexid))
  {
    Error("CENTERS index is not created!");
    return;
  }

  DSIndex centers(m_pDatabase, indexid);
  DSSet centertoc = centers.GetTOC();
  centertoc.Reset();

  while(centertoc.HasMore())
  {
    DSObject* elem = centertoc.Next();
    if(elem->IsKindOf(RUNTIME_CLASS(DSIndex)))
    {
      DSIndex* iptr = (DSIndex*)elem;
      DSXMLNode center = parent.AddChild("center");
      InsertIndexID(center,iptr->m_id);
      DSXMLNode centername = center.AddChild("name");
      centername.SetText(&iptr->m_strDesc);
    }
  }
  centertoc.CleanUp();
}
void DSBusinessLayerEMR::RequestDepartmentTOC(DSXMLNode &parent)
{
  DSIndexID indexid = GetIndexID(parent.FindNode("//indexid"));
  if(!m_pPlace->Dir()->Contains(indexid))
  {
    Error("Index does not exist!");
    return;
  }

  DSIndex departments(m_pDatabase, indexid);
  DSSet departmenttoc = departments.GetTOC();
  departmenttoc.Reset();

  while(departmenttoc.HasMore())
  {
    DSObject* elem = departmenttoc.Next();
    if(elem->IsKindOf(RUNTIME_CLASS(DSIndex)))
    {
      DSIndex* iptr = (DSIndex*)elem;
      DSXMLNode department = parent.AddChild("department");
      InsertIndexID(department,iptr->m_id);
      DSXMLNode departmentname = department.AddChild("name");
      departmentname.SetText(&iptr->m_strDesc);
    }
  }
  departmenttoc.CleanUp();
}
void DSBusinessLayerEMR::RequestCartTOC(DSXMLNode &parent)
```

```
{
  DSTime time = DSTime(CTime::GetCurrentTime());
  DSSet carttoc = m_Department.GetTOC();
  carttoc.Reset();

  while(carttoc.HasMore())
  {
    DSObject* elem = carttoc.Next();
    if(elem->IsKindOf(RUNTIME_CLASS(DSDocument)))
    {
      DSDocument* dptr = (DSDocument*)elem;
      if(dptr->m_strDesc != "_preferences")
      {
        DSXMLNode emr = parent.AddChild("document");
        DSDocument coverdata = DSDocument(m_pDatabase,dptr->m_ancestor.
     m_DocumentID);
        DSContents contents;

        if(coverdata.m_type.GetType() == DScopy)
          coverdata = DSDocument(m_pDatabase,coverdata.m_ancestor.m_DocumentID)
     ;

        if(dptr->m_ancestor.m_DocumentID.GetEditionID() != coverdata.
     m_strNewestEditionId)
        {
          // A newer edition is available
          DSDocumentID newcovdat = DSDocumentID(coverdata.m_id.GetPlaceID(),
     coverdata.m_id.GetGroupID(),coverdata.m_strNewestEditionId);
          coverdata = DSDocument(m_pDatabase,newcovdat);

          m_pCommands->GetDocFromDir(m_PersonID,time,m_DepartmentID,dptr->m_id)
     ;
          m_pCommands->RemoveDoc(m_PersonID,time,dptr->m_id);

          bool getsuccess = false;
          while(!getsuccess)
          {
            try
            {
              m_pCommands->GetDosFromDir(m_PersonID,time,m_ArchiveID,coverdata.
     m_membership.GetDossierID());
              getsuccess = true;
            }
            catch(DSError e)
            {
              if(e.GetType() != ERR_IDX_DOES_NOT_CONTAIN_DOS) throw e;
              else Sleep(200);
            }
          }

          DSDocumentID newref = m_pCommands->Copy(m_PersonID,time,coverdata.
     m_id);
          m_pCommands->ReturnDos(m_PersonID,time,coverdata.m_membership.
     GetDossierID());
          m_pCommands->PutDocInDir(m_PersonID,time,m_DepartmentID,newref);

          InsertDocumentID(emr,newref);
          contents = coverdata.GetContents();
        }
        else
        {
          // The current edition is up-to-date
```

```
          InsertDocumentID(emr,dptr->m_id);
          contents = dptr->GetContents();
        }

        emr.AddChild("firstname").SetText(&contents.Get()->FindNode("//_fornavn").GetText());
        emr.AddChild("lastname").SetText(&contents.Get()->FindNode("//_efternavn").GetText());
        emr.AddChild("cpr").SetText(&contents.Get()->FindNode("//_cpr").GetText());
        emr.AddChild("bed").SetText(&contents.Get()->FindNode("//_sengenr").GetText());
        CString no_of_drafts = contents.Get()->FindNode("//_antalkladder").GetText();
        if(atoi(no_of_drafts) > 0) emr.AddChild("draft").SetText(&no_of_drafts);
        else emr.AddChild("draft").SetText(&CString(""));

        contents.CleanUp();
      }
    }
  }
  carttoc.CleanUp();
}
void DSBusinessLayerEMR::RequestPersonTOC(DSXMLNode &parent)
{
  DSTime time = DSTime(CTime::GetCurrentTime());
  DSSet persontoc = m_Person.GetTOC();
  persontoc.Reset();

  while(persontoc.HasMore())
  {
    DSObject* elem = persontoc.Next();
    if(elem->IsKindOf(RUNTIME_CLASS(DSDocument)))
    {
      DSDocument* dptr = (DSDocument*)elem;
      if(dptr->m_strDesc != "_preferences")
      {
        DSXMLNode document = parent.AddChild("document");
        DSDocument coverdata = DSDocument(m_pDatabase,dptr->m_ancestor.m_DocumentID);
        DSContents contents;

        if(coverdata.m_type.GetType() == DScopy)
          coverdata = DSDocument(m_pDatabase,coverdata.m_ancestor.m_DocumentID);

        if(dptr->m_ancestor.m_DocumentID.GetEditionID() != coverdata.m_strNewestEditionId)
        {
          // A newer edition is available
          DSDocumentID newcovdat = DSDocumentID(coverdata.m_id.GetPlaceID(),coverdata.m_id.GetGroupID(),coverdata.m_strNewestEditionId);
          coverdata = DSDocument(m_pDatabase,newcovdat);

          m_pCommands->RemoveDoc(m_PersonID,time,dptr->m_id);

          bool getsuccess = false;
          while(!getsuccess)
          {
            try
            {
```

```
              m_pCommands->GetDosFromDir(m_PersonID,time,m_ArchiveID,coverdata.m_membership.GetDossierID());
              getsuccess = true;
            }
            catch(DSError e)
            {
              if(e.GetType() != ERR_IDX_DOES_NOT_CONTAIN_DOS) throw e;
              else Sleep(200);
            }
          }

          DSDocumentID newref = m_pCommands->Copy(m_PersonID,time,coverdata.m_id);
          m_pCommands->ReturnDos(m_PersonID,time,coverdata.m_membership.GetDossierID());

          InsertDocumentID(document,newref);
          contents = coverdata.GetContents();
        }
        else
        {
          // The current edition is up-to-date
          InsertDocumentID(document,dptr->m_id);
          contents = dptr->GetContents();
        }

        document.AddChild("firstname").SetText(&contents.Get()->FindNode("//_fornavn").GetText());
        document.AddChild("lastname").SetText(&contents.Get()->FindNode("//_efternavn").GetText());
        document.AddChild("cpr").SetText(&contents.Get()->FindNode("//_cpr").GetText());
        document.AddChild("bed").SetText(&contents.Get()->FindNode("//_sengenr").GetText());
        CString no_of_drafts = contents.Get()->FindNode("//_antalkladder").GetText();
        if(atoi(no_of_drafts) > 0) document.AddChild("draft").SetText(&no_of_drafts);
        else document.AddChild("draft").SetText(&CString(""));

        contents.CleanUp();
      }
    }
  }
  persontoc.CleanUp();
}
void DSBusinessLayerEMR::RequestJournalTOC(DSXMLNode &parent)
{
  // Determine the correct emr based on the reference document (ancester + membership)
  DSDocument emrref = DSDocument(m_pDatabase,GetDocumentID(parent.FindNode("documentid")));

  if(emrref.m_type.GetType() == DScopy)
    emrref = DSDocument(m_pDatabase,emrref.m_ancestor.m_DocumentID);

  DSDossier emr = DSDossier(m_pDatabase,emrref.m_membership.GetDossierID());

  DSSet emrtoc = emr.GetTOC();
  emrtoc.Reset();

  while(emrtoc.HasMore())
```

```
{
  DSObject* elem = emrtoc.Next();
  if(elem->IsKindOf(RUNTIME_CLASS(DSDocument)))
  {
    // if coverdata is requsted return only this document
    DSDocument* dptr = (DSDocument*)elem;
    if(parent.FindNode("subdossier").GetText() == "coverdata" && dptr->
   m_strDesc == "_coverdata" )
    {
      DSXMLNode document = parent.AddChild("document");
      InsertDocumentID(document,dptr->m_id);

      DSContents contents = dptr->GetContents();
      DSDocument firstedition(m_pDatabase,DSDocumentID(dptr->m_id.GetPlaceID
   (),dptr->m_id.GetGroupID(),"1"));

      document.AddChild("type").SetText(&dptr->m_strContentsType);
      document.AddChild("time").SetText(&firstedition.m_time.ToString());

      DSPerson creator(m_pDatabase,firstedition.m_creator);
      document.AddChild("creator").SetText(&creator.m_strName);
      document.AddChild("draft").SetText(&contents.Get()->FindNode("//_kladde
   ").GetText());

      contents.CleanUp();
      break;
    }
  }
  if(elem->IsKindOf(RUNTIME_CLASS(DSDossier)))
  {
    // return the contents of the designated subdossier in the emr
    DSDossier* dptr = (DSDossier*)elem;
    if(parent.FindNode("subdossier").GetText() == dptr->m_dossdesc.
   GetDescription())
    {
      DSSet emrsubtoc = dptr->GetTOC();
      emrsubtoc.Reset();

      while(emrsubtoc.HasMore())
      {
        DSObject* elem = emrsubtoc.Next();
        if(elem->IsKindOf(RUNTIME_CLASS(DSDocument)))
        {
          DSDocument* dptr = (DSDocument*)elem;
          DSXMLNode document = parent.AddChild("document");
          InsertDocumentID(document,dptr->m_id);

          DSContents contents = dptr->GetContents();
          DSDocument firstedition(m_pDatabase,DSDocumentID(dptr->m_id.
   GetPlaceID(),dptr->m_id.GetGroupID(),"1"));

          document.AddChild("type").SetText(&dptr->m_strContentsType);
          document.AddChild("time").SetText(&firstedition.m_time.ToString());

          DSPerson creator(m_pDatabase,firstedition.m_creator);
          document.AddChild("creator").SetText(&creator.m_strName);
          document.AddChild("draft").SetText(&contents.Get()->FindNode("//
   _kladde").GetText());

          contents.CleanUp();
        }
      }
```

```
        emrsubtoc.CleanUp();
        break;
      }
    }
  }
  emrtoc.CleanUp();
}
```

Appendix M

Article

Det Digitale Danmark – Effektivisering eller illusion?

Arbejder du i offentlig eller privat administration? Bruger du det meste af dagen på at dokumentere dit eller andres arbejde? Problemer med at finde det nyeste word-dokument på computeren? Er det nye dokumentsystem ubrugeligt?

Alt for mange dokumentsystemer leveres – i kundens øjne – med fejl. Nogle kan være forårsaget af fejlprogrammering men de fleste skyldes kommunikationsproblemer mellem leverandør og kunde – der er blevet leveret en hund men kunden ville have en kat. Kommunikationsproblemer og manglende forståelse for hinandens arbejdsområder og arbejdsrutiner er den primære årsag til fejl og resultaterer i tunge eller ubrugelige systemer. Løsningen er at forbedre forståelsen imellem parterne ved at finde et fælles grundlag – et sprog som begge taler og kan relatere til – og som kan udtrykke enhver arbejdsrutine og håndtere alle slags dokumenter.

Forståelsesproblemer mellem leverandøren og kunden kan oftest spores ned til manglende indsigt i det som indenfor softwareudvikling populært betegnes som 'domænet'. Enhver person som arbejder indenfor og har erfaring med et givet arbejdsområde er specialist i arbejdsområdet og dets arbejdsrutiner – specialist i domænet. Oveordnet set, er et domæne et genstandsområde hvori personer kan befinde sig, f.eks. arbejdspladsen, hjemmet eller indkøbscenteret. Det er med andre ord et afgrænset område man kan beskrive ved at observere det.

Indenfor et domæne gælder et sæt spilleregler, som ikke altid er lige lette at gennemskue. Et eksempel på et dokument-domæne er politik, nærmere bestemt Christiansborg. At beskrive Christiansborg udefra kan være forholdsvist nemt. At nedfælde de – måske uhensigsmæssige og ineffektive, men funktionelle – mekanismer, uskrevne regler og arbejdsrutiner som finder sted er langt mere kompliceret og kræver god indsigt i domænet. Fornuften i dem kan være svær at se for udenforstående netop pga. manglende indsigt domænet – det er med andre ord utænkeligt at en person uden erfaring med politik vil kunne nedfælde disse spilleregler. Det kræver een eller flere medarbejdere fra Christiansborg –

en eller flere domæne-specialister. Skal et informationssystem laves til Christiansborg kan en udvikler ved interviews af domæne-specialisten – kunden – forsøge at danne sig et overblik over domænet, men det kan besværes af kommunikationsvanskeligheder mellem de to parter – de har ikke samme opfattelse af hvad tingene betyder. Desuden kan det, som eksemplet demonstrerer, være svært for en udenforstående at få det komplette overblik og indsigt.

Kommunikationsvanskelighederne og mangel på erfaring med et arbejdsområde for udvikleren vil afspejles i udviklingsforløbet. Kunden kan ikke vide hvilke informationer der er vigtige at fortælle og udvikleren kan ikke spørge til problemstillinger der er ukendte for ham. Det kan lede til overraskelser når udviklingsprocessen afslører større mangler sent i forløbet. Ofte vil et utilstrækkeligt fundament betyde store forsinkelser og måske endda dårlige systemer til irritation for begge parter. AFs Amanda-system er et godt eksempel på misforståelser. Systemet var godt til bestemte arbejdsrutiner, bare ikke de arbejdsrutiner som de fleste AF medarbejdere benyttede. Domænet var ikke blevet analyseret til bunds og man forsøgte at strømline arbejdsrutiner, dvs påtvinge medarbejderne nye, men desværre ubrugelige, rutiner. Resultatet taler for sig selv.

Først når kommunikationsproblemerne mellem udvikleren og kunden er afhjulpet, og grundig indsigt i domænet er opnået kan udvikleren koncentrere sig om bruge sin teknologiske indsigt til at udtænke hvordan digitaliseringen af dokumenter bedst kan bruges til at lette arbejdsgangen og hvordan kundens øvrige behov bedst imødekommes. F.eks. vil de fleste kunder idag have XML i deres produkter, men XML i sig selv er ikke løsningen på noget, det er et redskab som en køkkenkniv. Brugt korrekt kan den blive uundværlig, men omvendt kan den også skabe flere problemer hvis brugt forkert. Udvikleren kan først tage stilling til om XML overhovedet er relevant og hvordan det benyttes hensigtsmæssigt når der er god forståelse for problematikken.

Det er vores holdning at problemet med manglende forståelse for domænet kun kan mindskes ved at lette samspillet mellem udvikler og kunde. Den største hindring er kommunikationsproblemerne, som opstår fordi parterne ikke taler samme sprog. Kunden fokuserer på de arbejdsrutiner i virksomheden som skal digitaliseres, imens udvikleren fokuserer på de teknologier og softwareudviklingsmetoder som skal anvendes i den sammenhæng. Oftest mødes de på halvvejen i form af grafiske software specifikationssprog, hvor man med pile, kasser og tændstikmænd skal udtrykke de nuværende arbejdsrutiner og den retning man ønsker at bevæge sig mod i det nye system. Problemet er, at dette stiller krav til kunden om skulle udtrykke sine arbejdsvaner i et ukendt overordnet sprog, hvor tvetydigheder, misforståelser og mangel på information ofte kan finde sted.

I stedet for at indordne sig under abstrakte udviklingsmetoder bør der istedet skabes et fælles sprog som begge kan forholde sig til og udtrykke sig i. Det er oplagt at basere dette på de praktiske erfaringer, som alle der har arbejdet med papirdokumenter kender til. Enhver person som arbejder med dokumenter har sin egen opfattelse af hvad et dokument er. På tværs af personer har disse opfattelser et sæt grundlæggende fællestræk som har været de samme de sidste mange hundrede år og derfor også stadigvæk præger nutidens dokumentbaserede arbejdsrutiner. Et papir-dokument kan kun befinde sig ét sted i verden ad gangen og kan kan være enten en original, kopi eller version – sidstnævnte opstår efter at man har rettet i en original eller kopi. Selvom et dokument er en kopi af et andet er det stadig to forskellige dokumenter uden synlig forbindelse andet end at indholdet er det samme. Man udfører ting såsom at kopiere, sende, underskrive og rette dokumenter man besidder. Et dokument har indhold som kan være hvad som helst: tekst, skemaer, tegninger, fotografier, grafer, billeder med meget andet. Patientjournaler, kontrakter, arkitekttegninger, spørgeskemaer og fagforeningspapirer er derfor dokumenter med de nævnte fællestræk. Disse fundamentale principper og egenskaber for papirbaserede dokumenter kan overføres til computere – nye teknologier såsom digitale signaturer er påkrævet, men det kan lade sig gøre.

Løsningen på kommunikationsproblemerne er derfor at skabe et nyt dokumentorienteret udviklingssprog, som er baseret på de fundamentale dokumentprincipper, og som oversætter disse til deres digitale ækvivalent. Dette sprog vil udgøre et sæt byggeklodser, som alle arbejdsrutiner indenfor dokumenthåndtering er sammensat af. Der kan altså dannes et fælles grundlag, hvorfra alle dokumentdomæner kan udtrykkes – med begreber man har været vant til fra sin dagligdag. Med udgangspunkt i et sådant sprog reduceres et indledende udviklingsforløb til beskrivelser af hvad man gør med sine papirer og hvordan de ser ud. Andre – mindre interessante, men nødvendige – emner såsom sikkerhed, systemarkitektur, distribution af information, XML m.m., kan holdes udenfor udviklingsarbejdet da disse ting kan realiseres overordnet helt uafhængigt af domænets arbejdsrutiner.

Med udgangspunkt i et sådant sprog er det muligt at indledningsvist at fastholde eksisterende arbejdsrutiner ved digitaliseringen. Når brugerne har vænnet sig til at arbejde elektronisk kan arbejdsrutiner omlægges ved at udtrykke dem i det underlæggende sprog istedet for at omvæltningen sker fra første dag. På den måde tvinges udvikleren til sætte sig ind i den eksisterende måde at gøre tingene på i stedet for at der udvikles helt nye – man kan ikke udvikle nye rutiner på baggrund af ingenting. Et nyt dokumentsystem skal derfor kunne understøtte eksisterende arbejdsrutiner – måske er rutinerne uhensigtmæssige, men de fungerer. Det gør det lettere at udvikle et system tilpasset til brugerne og ikke omvendt. Tankegangen skal være, at et nyudviklet system som minimum bør kunne efterligne de eksisterende arbejdsrutiner – man kan ikke gen-tænke før man kan sætte sig ind i, forstå og efterligne det eksisterende. Nye rutiner kan indarbejdes gradvist – disse er noget af det sværeste at ændre og det bør ikke foregå fra den ene dag til den anden.

Det er netop i denne tid at kimen lægges til de fremtidige digitale dokumentsystemer. Det sker i takt med indførelse eller sammenlægning af eksisterende systemer overalt. Der arbejdes i øjeblikket på introduktion af elektroniske patient journaler (EPJ) på hospitalerne, og for nylig indgik tre større firmaer en rammeaftale om levering af en fælles offentlig standard for sags- og dokumenthåndtering til kommunerne (FESD) – ambitiøse IT-projekter til flere milliarder. De skal især være opmærksomme på den ovenstående problemstilling da der tale om omfattende domæner med mange mennesker og mange komplekse arbejdsrutiner, som skal digitaliseres og strømlines med tiden. Der er i denne forbindelse adskillige faldgruber inden målet er nået og sandsynligheden for at man falder i afhænger direkte af forståelsen af domænet.

Det er derfor vigtigt netop nu at tage stilling til ovenstående problemer, tænke fremad og ikke 'nøjes' med forjagede midlertidige løsninger. Man stiller generelt kritiske krav til så meget andet. Hvorfor ikke også stille krav til at et af de vigtigste arbejdsredskaber fungerer optimalt for den enkelte? Det er kun rimeligt at denne udvikling sættes i gang nu og udfordrer software firmaerne. Selvfølgelig skal virksomhedens arbejdsrutiner diktere IT-systemet og ikke omvendt. Selvfølgelig skal systemet kunne tale XML. Selvfølgelig skal nutidens teknologier udnyttes til at lette og effektivisere den eksisterende arbejdsgang. Alle kan blive enige om disse punkter, men alligevel opstår der stadig problemer netop i disse sammenhænge. En manglende forståelse af domænet og dets spilleregler er ofte roden til problemerne og det er derfor vigtigt at basere de fornuftige krav på en grundig domæneanalyse.

XML

XML er en enkel og struktureret måde at beskrive information på, hvilket har gjort den særdeles anvendelig når information skal udveksles. XML er baseret på en mere end 20 år gammel standard, hvis principper daterer helt tilbage til 1960'erne. Siden da har disse principper reelt ikke ændret sig, hvilket viser styrken i fundamentet og understreger at XML er kommet for at blive. Der er med andre ord ikke tale om 'ny' teknologi, som vil blive erstattet med tiden.

Digital signatur

Ligesom med 'virkelige' underskrifter kan en digital signatur bruges til at finde ud af hvem der har skrevet under på en tekst. Den mest udbredte metode til at lave digitale signaturer er via et personligt digitalt certifikat. Det indeholder information som sætter ejeren i stand til at digitalt signere og sende hemmeligheder over internettet – alt hvad der nødvendigt for at opretholde privatlivet på nettet.

Man kan få udstedt et gratis certifikat fra den danske stat gennem TDC, som har problemer med at udbrede kendskabet til de statsfinansierede certifikater. Det kan tilskrives manglende information og reklame for konceptet, samt de begrænsede anvendelsesområder, som dog er på hastig fremmarch. Desuden er certifikaterne stadig behæftet med flere problemer såsom besværlig installation og at Windows-maskiner ikke per automatik genkender TDC certifikater – problemer som kunne være undgået fra starten men det kan dog stadig nås. Ideen med at udstyre samtlige borgere med certifikater er god og kan være med til at reducere og på sigt fjerne alt hvad der hedder uønsket email – spam – og virus.

For digitale underskrifter gælder det at man ikke kan ændre teksten uden at underskriften bliver ugyldig og underskriften kan ikke forfalskes – så på den måde er den stærkere end underskrifter på papir. Rent teknisk er en digital signatur data, som kan vedhæftes en tekst. Modtageren af teksten kan udfra vedhæftede data afgøre hvem der har skrevet under og om teksten er intakt. Man kan måske være tilbøjelig til stadig at stole mere på normale underskrifter, men faktisk er de digitale signaturer så sikre, at det er muligt at bruge dem til at foretage sig ting, som tidligere krævede underskrevede papirer og/eller fremmøde – eksempelvis det at udfylde sin selvangivelse. Derfor er det naturligvis nødvendigt at vise samme påpasselighed med sit digitale certifikat som man gør med sit dankort og sin homebanking.

Appendix N

Business Plan

Business Idea

We offer a document oriented framework, capable of managing any kind of information – any kind of document. The framework supports concepts like version tracking, encrypted XML data exchange, extensive security settings on the individual document and distribution.

The framework differentiates itself from existing products by offering a series of building blocks – a series of denominators – that all document oriented business processes can be broken into. The framework can thereby imitate and support any – perhaps ineffective – but working business process. The building blocks and their interaction can be expressed in a simple scripting language originating in the domain of documents therefore using terminologies easy to understand. Using the language the end-user is capable of expressing own business processes effectively minimizing problems arisen from communication problems between developer and end-user that are common these days.

The philosophy behind the product is that the customer knows what is best for him and he should therefore be able to express needs in ways not unfamiliar to him. It is also essential that tailoring can be done in every possible way – the framework should not place restrictions on the flexibility. It is imperative to be able to offer adoption of existing business process, despite them being ineffective – re-engineering is easy carried out in due time by tailoring a new set of rules in the scripting language and introducing new versions of documents structures.

The overall framework provides a solid foundation for tailored future proof systems for companies in need of user-friendly, but effective ways to cope with and manage their documents. This includes, among others, pharmaceutical companies, construction companies, such as Sund & Bælt (Femern Bælt), public case management (FESD) and the hospitals that are introducing electronic medical records (EPJ).

How to Profit

The product can be a

- **Turn-key solution** – sold off as a complete system to a customer, who in cooperation with us tailor the framework to specific needs.
- **Off-the-shelf product solutions** (OTS) – The framework is tailored to support a specific generic kind of need (such as management of Microsoft Word documents) and sold as an OTS product.
- **Framework solution business-to-business** (b2b) – sold as a framework to a customer, who tailors and sells it to one of his customers.

The customers can be roughly divided into the following segments:

- **Large scale customers** – OTS solutions. Many competitors with well-established systems.
- **Large scale customers** – turn-key solutions. Many competitors, but many produce inferior products (explained in the market analysis).
- **Medium scaled customers** – OTS solutions. Several competitors.
- **Medium scaled customers** – turn-key solutions. Many competitors but many produce inferior products (explained in the market analysis).
- **b2b** – Framework solution. Zero or few competitors at the medium scaled customer level.

We will target the turn-key solution group + b2b as it is here the product and the philosophy behind will supersede existing products. As a secondary market we expect to produce selected OTS low-cost solutions that will honor low and medium scaled customers, their budget and needs – we expect this to be a simple extension of the original product and would therefore not require anything particular to realize.

Market Analysis and Market Strategy

The market of document systems is saturated, but we believe that our approach is a new way of addressing the problem. We believe it is more appealing to customers and ultimately results in less expensive systems as the production time is minimzed.

The market leader today of document software – Documentum – targets large-scale customers, which is reflected in their prize range starting from $1.000.000. They try to approach the customization market with new products intended to be customized by the customer via user friendly GUIs. This is a trade-off between flexible systems vs. the customer being able to do the customization himself – the more flexible the more options for the user, thereby inadvertently preventing the ordinary user from understanding the customization process. We believe that customization is best handled in cooperation between a skilled developer and a skilled domain specialist – the customer. No matter what the tendencies are in existing document software this symbiosis will never vanish. If customer employees achieve expertise in complex scripting languages, GUIs etc, it would result in that the specialized employee would alienate himself from other employees (would go from domain specialist to developer) thereby re-introducing the common and infamous communication problems.

Other – in comparison to the market leader – inferior products targeted at low- and mid scaled customers are not very user-friendly nor effective, which leads to believe that a fast customization of our system could compete with their products.

There are many companies that can offer turn-key document management systems, and some excel in doing this. It is, however, often the conclusion that the systems are inadequate and do not comply with what was intended. It is our conjecture that it is due to inferior development methods and a lack of domain understanding. Our product and the philosophy behind deals with these problems in an intuitive methodological way. Our framework provides a scripting language that both developer and customer can understand and relate to while hiding the technical aspects such as security, XML and distribution. The result is better and faster development methods implicitly leading to better products for less money.

To our knowledge there are no 'true' document oriented frameworks at the medium scaled customer level. Some companies claim that their product can be tailored to the customer but to some degree they enforce standard business processes and limitations on the customer disqualifying them from being considered frameworks. Ultimately the customer adapts – not the product. The b2b market will be difficult to penetrate as it requires several success stories before third party companies will be willing to purchase the framework. It is, however, reasonable to believe that a couple of success stories could make it a market. It would then compare to the Navision product of Microsoft Business Solutions and the many solution centers that excel in doing customization of the general product.

Considerations on IPR

We will not be able to patent the concepts as they are, despite their neglected use, well-known development principles. The product itself has to be accompanied by the philosophy of domain engineering which can be protected to some degree. We will not expect to have competitors with the same functionality and principles as it would require a complete new start for all the existing players in their basic design. New companies would not be able to copy the concept immediately as it would require a couple of years of development.

Business Model

We intend to find as many customers as possible in order to mature and refine the product. This arrangement would initially be free of charge for the customers doing the beta testing except for salaries for the involved developers. As the product does not have infinite growth potential the company would gradually transform to consultancy work and modifications and maintenance of existing systems.

(Current) Competencies

Our team consists of two, soon-to-be, Masters of Science and Engineering in the area of software development. We have researched the document domain during the last 12 months which has given us a profound understanding of the domain and the problems within this area. Academically, we have years of experience with all the fields of software engineering, such as analysis, design, and implementation. Both of us have worked for years in IT service and consultancy, strengthening our customer oriented skills while observing the general behavior and problems of 'normal' IT users. These skills and experiences provide us with a clear understanding of the needs of users and the ability to meet their demands in regards to software development.

Managerial and Organizational Setup

The organization we expect to initially pursue is a small company, with a qualified CEO managing the administrative aspects, such as acquiring customers. We expect to be part of the R&D together with a couple of 'coders' – it will our job to refine the framework and in time customize the framework for newly acquired customers.

Bibliography

[1] ODA (Office Document Architecture): What is it? What is it good for? *The Seybold Report on Publishing Systems*, 19(7):3–5, 1989.

[2] Documentum a division of EMC Corporation. *Documentum.* www.documentum.com.

[3] Martín Abadi, Michael Burrows, and Butler Lampson. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15:706–734, 1993.

[4] Gregory R. Andrews. *Multithreaded, Parallel, and Distributed Programming.* Addison Wesley Longman, Inc., 2000.

[5] Brian Berliner. Concurrent versions system - CVS, 1989.

[6] Dines Bjorner. *The SE Book*, volume 3. 2003.

[7] Ronald Bourret. *XML and Databases.* 2003.

[8] Lou Burnard. What is SGML and How Does it Help? *Modelling Historical Data: Towards a Standard for Encoding and Exchanging Machine-Readable Texts*, pages 65–79, 1991.

[9] Ian R. Campell-Grant. Introducing ODA. *Computer Standards and Interfaces*, 11:149–157, 1991.

[10] Ian R. Campell-Grant and Krõnert Günther. First Implementation of The ODA Standard. *Information Processing 89 - Proceedings of The IFIP 11th World Computer Congress*, pages 1025–1028, 1989.

[11] Steen Dawids, Allan Engquist, and Jacob Rosenberg. *Memo Medica.* Munksgaard Danmark, 2001.

[12] EPJ-Observatoriet. *Statusrapport 2003.* EPJ-Observatoriet, 2003.

[13] Gert Galster. Begrebsmodellen i G-EPJ. 2003.

[14] Michael Gertz. *Oracle/SQL Tutorial.* Database and Information Systems Group, Department of Computer Science, University of California, Davis, 2000.

[15] GESI. *Distributed Healthcare Environment.* www.gesi.it.

[16] Raise Language Group. *The RAISE specification language.* Prentice Hall, 1992.

[17] Amager Hospital. *Manual for fællesjournal.* Amager Hospital, 2001.

[18] R. Hunter, P. Kaijser, and F. Nielsen. Oda: A document architecture for open source systems. *Computer Communications*, 12(2):69–79, 1989.

[19] Christian Krog Madsen, Rasmus Dyhrberg, and Nikolaj Christensen. Specifikation af elektroniske patientjournaler. Technical report, IMM, DTU, December 2002.

[20] Allan Lindqvist, Brian Christensen, Mads Johnsen, and Søren Risgård. Course 02262: The Document System. Technical report, IMM, DTU, July 2003.

[21] Rolf Molich. *Brugervenligt webdesign.* Nyt Teknisk Forlag, 2003.

[22] Danish Ministry of Health. *Handlingsplan for Elektroniske Patientjournaler.* Danish Ministry of Health, 1996.

[23] Danish Ministry of Health. *National strategi for IT i sygehusvæsenet.* Danish Ministry of Health, 1999.

[24] Danish Ministry of Justice. Act on Processing of Personal Data. *Official Journal*, June 2000.

[25] Indenrigs og Sundhedsministeriet. *National IT-strategi for sundhedsvæsenet.* Indenrigs og Sundhedsministeriet, 2003.

[26] Steve Price. Oda: The iso standard for electronic document interchange. *International Open Systems 88*, pages 337–354, 1988.

[27] Jeff Prosise. *Programming Windows with MFC.* Microsoft Press, 1999.

[28] Airi Salminen and Frank Wm. Tompa. Requirements for XML Document Database Systems. *Proceedings of the ACM Symposium on Document Engineering*, pages 85–94, 2001.

[29] Torben V. Schroder, Svend Schulze, Jannik Hilsted, and Jan Aldershvile. *Basisbog i Medicin & Kirurgi.* Munksgaard Danmark, 2004.

[30] Robin Sharp. *Principles of Protocol Design.* DTU-TRYK, draft second edition, August 2002.

[31] Ian Sommerville. *Software Engineering.* Addison-Wesley, sixth edition, 2001.

[32] William Stallings. *Cryptography and Network Security.* Prentice-Hall, second edition, 1999.

[33] Douglas R. Stinson. *Cryptography - Theory and Practice.* CRC Press, 1995.

[34] Sundhedsstyrelsen. *Grundstruktur for Elektronisk Patient Journal.* Sundhedsstyrelsen, 2001.

[35] Sundhedsstyrelsen. *Sundhedsstyrelsens klassifikation af personale i sundhedssektoren*. Sundhedsstyrelsen, 2002.

[36] Sundhedsstyrelsen. Komponenter i G-EPJ - på begrebsniveau. 2003.

[37] Hovedstadens Sygehusfælleskab. *Procesbeskrivelser*. 2001.

[38] Systematic. *Columna Open Architecture*. www.systematic.dk.