Desktop Integration with a Web Based Application

By

Johan Gustafsson

LIU-IDA/LITH-EX-A—12/014—SE

2012-04-17
Final Thesis

Desktop Integration with a Web Based Application

By

Johan Gustafsson

LIU-IDA/LITH-EX-A—12/014—SE

2012-04-17

Supervisor: Jesper Strige, Ipendo Systems
Examiner: Simin Nadjm-Tehrani
Abstract

This master thesis work was done at Ipendo Systems in Linköping, a company that makes an intellectual property management system called Ipendo Platform.

The master thesis describes the design and development of an extension to a web based solution to work as desktop application and demonstrating the solution with an Outlook plugin. The goal was to improve the workflow for the user when handling documents received by mail and also find and evaluate a model for product integration that could be re-used for future projects.

The result of the master thesis is an Outlook plugin and a web service that exposes part of Ipendo Platform functionality in a service layer. As a final test the solution was tested in a production environment to simulate real world usage.

The report provides conclusions about the pros and cons of this kind solution and how the current design and implementation of Ipendo Platform has affected the outcome.
Contents
Abstract .................................................................................................................................................. ii
1 Introduction ......................................................................................................................................... 1
  1.1 Background .................................................................................................................................... 1
  1.2 Ipendo Group, Intellectual Property Management ....................................................................... 1
  1.3 Problem definition .......................................................................................................................... 2
  1.4 Objective ......................................................................................................................................... 2
  1.5 Product requirements .................................................................................................................... 3
    1.5.1 Product Functional Requirement ............................................................................................. 3
    1.5.2 Constraints ............................................................................................................................... 3
    1.5.3 Interfaces .................................................................................................................................. 4
    1.5.4 User characteristics .................................................................................................................. 4
    1.5.5 Assumptions and Dependencies ............................................................................................. 4
  1.6 Approach ....................................................................................................................................... 4
    1.6.1 Prototyping (pre-study) ........................................................................................................... 4
    1.6.2 Design ....................................................................................................................................... 4
    1.6.3 Implementation ....................................................................................................................... 4
    1.6.4 Testing ...................................................................................................................................... 5
    1.6.5 Evaluation and discussion ....................................................................................................... 5
  1.7 Limitations of scope ....................................................................................................................... 5
  1.8 Chapter Overview and Reading instructions ................................................................................ 5
2 Ipendo Platform .................................................................................................................................. 6
  2.1 Intellectual Property ..................................................................................................................... 6
    2.1.1 Copyright ................................................................................................................................... 6
    2.1.2 Patent ....................................................................................................................................... 6
    2.1.3 Trademark ................................................................................................................................. 8
    2.1.4 Design ....................................................................................................................................... 8
  2.2 Terminology of Ipendo Platform ................................................................................................. 9
    2.2.1 Nodes ....................................................................................................................................... 9
    2.2.2 Cases and Families .................................................................................................................. 9
    2.2.3 Matters .................................................................................................................................... 9
    2.2.4 Events ..................................................................................................................................... 9
2.2.5  Documents ........................................................................................................ 10
2.3  Uploading a document to a case........................................................................... 10
2.4  Ipendo Platform Design & Architecture .................................................................. 12
   2.4.1  Multi-layer architecture ................................................................................... 12
   2.4.2  Current state and architectural view of Ipendo Platform .................................. 13
   2.4.3  Database configuration ..................................................................................... 15
2.5  Security.................................................................................................................... 15
   2.5.1  Technology used ............................................................................................... 15
3  Service Oriented Architecture...................................................................................... 17
   3.1  Service Design paradigm ...................................................................................... 18
      3.1.1  Standardized Service Contract .................................................................... 18
      3.1.2  Service Loose Coupling ............................................................................. 18
      3.1.3  Service Abstraction ...................................................................................... 18
      3.1.4  Service Reusability ....................................................................................... 18
      3.1.5  Service Autonomy ....................................................................................... 19
      3.1.6  Service Statelessness .................................................................................... 19
      3.1.7  Service Discoverability ............................................................................... 19
      3.1.8  Service Composability ............................................................................... 19
   3.2  SOA and Cloud computing .................................................................................... 19
4  Middleware .................................................................................................................. 21
   4.1  CORBA ................................................................................................................. 21
      4.1.1  Transport ...................................................................................................... 21
      4.1.2  Service Description ...................................................................................... 21
      4.1.3  Service Discovery ....................................................................................... 22
   4.2  Jini ........................................................................................................................ 22
      4.2.1  Transport ...................................................................................................... 22
      4.2.2  Service Description ...................................................................................... 22
      4.2.3  Service Discovery ....................................................................................... 22
   4.3  Web Services ....................................................................................................... 22
      4.3.1  Transport ...................................................................................................... 23
      4.3.2  Service Description ...................................................................................... 23
      4.3.3  Service Discovery ....................................................................................... 23
      4.3.4  REST ........................................................................................................... 23
      4.3.5  Windows Communication Foundation ....................................................... 24
Appendix B

Appendix A – Service Interface API .............................................................. 57

Appendix B – Uploading a document from Platform interface ...................... 60

11.1.1 Save attachments ........................................................................... 60

11.1.2 Login .............................................................................................. 61

11.1.3 Find destination ............................................................................ 61

11.1.4 Upload document (from family view) ........................................... 64

12 Appendix C –IPOP User Interface screenshots ........................................ 68

Appendix D - Use cases ............................................................................ 79

UC1 Login ..................................................................................................... 79
Description .............................................................................................................................................. 79
Actors .......................................................................................................................................................... 79
Primary Actor ............................................................................................................................................. 79
Preconditions ............................................................................................................................................. 79
Success Guarantee .................................................................................................................................... 79
Main Scenario ............................................................................................................................................ 79
User cancel login dialog ............................................................................................................................ 79
Second level authentication is enabled ...................................................................................................... 80
User cancel second level authentication .................................................................................................. 80
The “one-time” password expires ............................................................................................................. 80
First level authentication fails .................................................................................................................... 80
Too many login attempts ............................................................................................................................. 80
Second level authentication fails .............................................................................................................. 80
UC2 Upload attachment .............................................................................................................................. 81
Description .................................................................................................................................................. 81
Actors ........................................................................................................................................................... 81
Preconditions .............................................................................................................................................. 81
Success Guarantee ...................................................................................................................................... 81
Main Scenario ............................................................................................................................................. 81
User cancels the upload ............................................................................................................................... 82
The dialog closes ......................................................................................................................................... 82
Attachment is too large ................................................................................................................................ 82
1 Introduction

This report is the result of a master thesis project carried out at Ipendo Systems in Linköping, September 2009 – January 2010. The master project is a partial fulfillment of a degree in Industrial Management and Engineering at Linköping University (30 credit points).

Chapter 1 describes the background and problem definition for the master thesis project.

1.1 Background

Management of intellectual property (IP) has become an important issue for many companies nowadays. Some companies have very large IP-portfolios and keeping track of them can be a tedious, costly and time consuming task, especially for a global company needing to register the same patent, trademark or other IP right in multiple countries.

The Ipendo platform is a completely web-based solution offered as “Software as a Service” (SaaS) and as an in-house installation. When offered as SaaS the server environment is hosted by Ipendo and is always accessible over Internet. Some of the largest customers have chosen to host their own environment but most customers seem to prefer the SaaS approach since it offers the best total cost of ownership and accessibility.

The fundamental goals of Ipendo platform are to:

1. Reduce costs by simplifying administration and contact with partners.
2. Give customers control over their IP.
3. Give customers a tool to analyze the portfolio in order to get an overview over the content and help the customer maximize profit from its IP Portfolio.
4. Provide a better understanding for IP among customer users.

Ipendo Platform simplifies IP administration by allowing customers to open up access to selected part of their portfolio to partners. The partners can then access the portfolio wherever they are.

For many companies inventions are very important and the source for creating revenue in the future. Without a simple and smooth filing system for potential inventions a lot of them may never be filed; which puts future revenue and business opportunities at risk. Ipendo Platform includes modules that simplify the process for inventors filing inventions to the IP department of the company. The inventors can add pictures and documents to their application and the IP department can audit the application and ask for additions from the inventor if needed. Inventors are also given the opportunity to follow the whole process; from an idea at an early stage, to an approved patent. This is one example of how Ipendo Platform is used as a collaboration platform [1].

1.2 Ipendo Group, Intellectual Property Management

Traditionally administration of IP has been done by the IP department of the company or by partners providing administrative services, e.g. patent bureaus. Many customers used Excel documents or other lists to keep track of due dates for applications and payments and some used software tools for organizing portfolio case data. The market for portfolio management has therefore traditionally consisted of service and software system providers [2].
Ipendo Group was founded around the business idea to offer a complete solution for IP management, both in terms of software tools for portfolio management and collaboration as well as administrative services. Ipendo aims to offer more than only case data management and have positioned the software as a “collaboration platform” rather than portfolio management software. Also services such as filing of patents and payment of annual fees are offered as a complement to Ipendo Platform [2].

1.3 Problem definition
IP management today includes keeping track of a lot of external documentation. Ipendo Platform therefore offers the functionality to upload and store a wide range of document formats. Some of the customers receive a lot of documents by mail, and the process of uploading it to the platform can be very time consuming, since the mail client is unable to directly communicate with the platform. Uploading a document received by mail means first to save it as a file on the local computer, open a web browser, login to the platform, find the destination and once again locate the saved document on the computer before uploading it to the platform. For some customers, who are dealing with a lot of documents on a daily basis, this workflow is too slow and time consuming.

The topic for this master thesis was suggested by Ipendo Systems, a subsidiary of Ipendo AB who is responsible for all software development and maintenance of Ipendo Platform, and is based on customer feedback wishing to improve document handling and mail client integration with their software solution.

Since this would be the first attempt to integrate a desktop application with Ipendo Platform, Ipendo Systems wanted the solution to be a re-usable strategy for integration with desktop applications and other external clients (e.g. mobile devices).

1.4 Objective
The master thesis should answer the following questions:

• What would be a suitable technology and architecture for the communication between the Ipendo Platform™ and an Outlook-plug-in?

• How should the solution be designed to make it secure and without risk of leaking confidential information?

• How should the plug-in be distributed to make installation and software updates seamless for the end-user?

• What lessons can be learned on integrating Ipendo Platforms with other applications? How could it be made easier?

The goal of this master thesis is to design a solution on how to integrate Ipendo Platform with a desktop application. The solution should then be applied by developing an Outlook-plug-in which is able to communicate and upload documents to Ipendo Platform.
Since Ipendo Systems in the future will have similar integration projects, the suggested solution should be evaluated from both a practical and theoretical standpoint so that parts of this “design pattern” can be re-used for other projects. Since this kind of integration has not been tried before with Ipendo Platform it will also give useful experience about the level of difficulty for integration and what needs to be improved in the product.

1.5 Product requirements
This section describes the product requirements for the Outlook plug-in as agreed upon with Ipendo Systems.

1.5.1 Product Functional Requirement
All functional requirements are listed below:

1. Log in to the Ipendo Platform with username and password. It must also handle second level authentication with one-time passwords. No other functionality should be accessible before a successful login. (REQ F1)

2. Show a tree-view of the objects accessible to the user and be able to select a destination to store the document from that view (REQ F2)

3. A search bar in which the user can enter a name or a property to find the object in the account on which the document will be stored. There should also be a search option to filter on “Family”, “Case” and “Matter”. (REQ F3)

4. Upload file(s) to the Ipendo platform after selecting destination from either the tree-view or the search bar result. Valid destinations should be “Families”, “Cases” and “Matters”. The user should be able to specify the document type before it is uploaded and require a confirmation by the user before execution in order to avoid feeding information to the wrong portfolio. (REQ F4)

5. Log out. This should end the session and disconnect the client’s network connection. (REQ F5)

Additional features (may be delayed to later versions):

6. Download documents from Ipendo Platform and add as an attachment (REQ F6)
7. Possibility to also upload mail content to Ipendo Platform (REQ F7)
8. Edit properties of a document (after upload) (REQ F8)

1.5.2 Constraints
The constraints define the non-functional requirement for the software.

9. Platform and application independent server interface. In other words it should be possible in the future to create clients for other applications and platforms, e.g. Lotus Notes and handheld devices. (REQ N1)
10. The transfer of files to the platform must be secure and encrypted. (REQ N2)
11. Updating the Outlook plug-in should be automatized and require minimal user interaction except on initial installation. (REQ N3)

12. Performance should be on a par with the previous solution and it should be able to upload attachments up to 30 MB within 1 minute (per attachment) (REQ N4)

1.5.3 Interfaces
The interface to the user will be a graphical user-interface included in the Outlook plug-in. All interaction between the software and the user is performed with mouse and keyboard.

1.5.4 User characteristics
The intended users are assumed to have appropriate training in IP management and the processes involved, but do not necessarily have a technical background. It is also assumed that the users are familiar with the Ipendo Platform web interface and the concepts used there.

1.5.5 Assumptions and Dependencies
The following software is required on the client computer in order run the plug-in:

1. Microsoft Windows

1.6 Approach
The work process of the master thesis was divided into

1. Prototyping (pre-study)
2. Design
3. Implementation
4. Testing
5. Evaluation and discussion

1.6.1 Prototyping (pre-study)
Since I had almost no experience working with Microsoft .NET framework and its development tools I first performed a literature study of the conceivable technologies and created a number of “proof of concept” prototypes in order to familiarize myself with the technology and to try out some ideas I had about the design of the software. The result of this can be found in the chapter “Pre-study & Important design Decisions”.

1.6.2 Design
The result of the pre-study was discussed with the supervisor and the project description was broken down to a more detailed requirements specification. After my supervisor had approved the requirements and use-cases, an architectural design for the software was created. The design and architecture are described in detail in the chapter “Design”.

1.6.3 Implementation
The implementation was done in two iterations, where the first iteration focused on finishing the infrastructure of the architecture, and the second iteration also integrated the new software with the Ipendo Platform. The design documents were also updated and refined at the end of each of the two iterations.
1.6.4 Testing
In order to make sure that the requirements had been met, test cases were written based on the use-cases, to test the functional requirement and non-functional requirements. The final test was to configure the system on a production environment in the US. The result of both functional and non-functional testing is discussed in the chapter “Testing”.

1.6.5 Evaluation and discussion
The project was evaluated from how well it met these criteria:

1. The improvement of workflow compared to the existing system.
2. How well the technical solution met the objectives set up in the beginning of the project.

1.7 Limitations of scope
The software developed is a prototype and some features that are not necessary to prove the concept will not be developed because of the limited time frame of the project. This mostly regards the Ipendo Platform access control policy and logging.

1.8 Chapter Overview and Reading instructions
Chapter 2 - Introduction to the basics of intellectual property and Ipendo Platform. The first section regarding IP can be skipped by readers already familiar with the subject. Knowledge about Intellectual Property is not needed for understanding the solution or this report but is interesting as a background to understand the problems that are solved by using Ipendo Platform.

Chapter 3 & 4 – The topics of these chapters are about service oriented architecture and about common middleware technologies. It can be skipped by readers already familiar with the subject.

Chapter 5 – This chapter deals with the result of the pre-study and design decision for the solution.

Chapter 6 – Details about the implementation and experiences learned during the development phase are presented in this chapter.

Chapter 7 – Describes the result of testing.

Chapter 8 – Is devoted to conclusions and lessons learnt.
2 Ipendo Platform
This chapter aims to describe Ipendo Platform, focusing on the parts most relevant for understanding this master thesis project. It starts with a short overview of the terminology used within the intellectual property field and Ipendo Platform, and continues with a detailed description of document uploading and the technical characteristics of the current system.

2.1 Intellectual Property
Since Ipendo Platform is a system for working with intellectual property (IP), this section gives a short introduction of intellectual property for the reader not familiar with the area. Most of the content of this section conforms with IP rights globally, but if not stated otherwise, it is based on the Swedish legislation. The four most common type of IP are copyright, patents, trademarks, and design patterns [3]:

2.1.1 Copyright
Copyright is the part of the IP right protecting music, literature and other original work. There is no application process for copyright; instead the author automatically receives protection when the work is created. Copyright gives the author the exclusive right to decide how their work is presented and distributed. This could therefore lead to a right to economic compensation for the author [4].

Work comprised in copyright law is written or spoken productions, computer software, databases, musical or theatrical compositions, artwork (including photography), architecture of buildings, and all other expression of creativity in literature or art. To receive protection it is required that the work created reaches the minimal standard of originality, and that the work should originate from the author’s personal and creative achievement [4].

The protection for the author is often marked by the symbol for copyright - ©. In Sweden however, since the work is automatically protected if it fulfills the minimal standard of originality, using the ©-symbol does not in itself create copyright and has no legal implication. Some authors still use the symbol to deter unlicensed use of their protected work [4].

2.1.2 Patent
Patents are legal documents specifying a technical invention. Patents are territorial and the same invention can therefore be patented in a number of countries. These patents usually have the same owner and are related to each other by their process of application. In that case they form a “patent family” [5].

The owner of a patent is given certain rights to exclude others from using an invention commercially. That is commercially using, selling, offering and keeping an invention in stock as specified in the claim section of the granted patent for the specific country. Since the patent law in differs between geographical jurisdictions, the scope of protection can vary between patents in the same family [5].

There are several requirements that a patent application need to fulfill in order to be approved [5]:

1. New: The invention for which the application regards must not earlier be publically known. This applies for the whole world, and for example a published report about the invention is enough for it to not count as a new invention.
2. Inventive: Means that the invention must substantially differ from already existing inventions.
3. Industrially applicable: The invention must be applicable in the industry, e.g. possible to manufacture.

There is no “world patent”, but many patent offices can handle application to multiple countries at the same time if an agreement exists between the countries. The two largest organizations for this is Patent Cooperation Treaty (PCT) with 130 member countries, and European Patent Office (EPO) that includes most of the European countries. A PCT application does not in itself grant a patent, but (slightly simplified) the right to proceed with a patent application within 30 month in the countries that signed the treaty. In other words it is a grace period under which an evaluation of different markets can take place and then the actual patent applications can be done only in the countries that seem interesting [5].

If an application to EPO is approved, it is usually automatically approved in all the EPO member countries. There administrative routines can differ some between the member countries, and some may require their own auditing in which the application might be denied. Also some parts of the patent needs to be translated to the local language and patents fees for each country must be paid in order to validate the patent in a specific country [5].

The cost for protecting a patent can vary greatly. In Sweden and in the US, a valid patent can be created for less than 5000 SEK. If an international protection is desired and the process is handled through a patent bureau with local representatives in respective country, the cost can exceed several millions. To keep a patent “alive” an annual fee must be paid and the fee increases for each passing year. In Sweden the fee starts at 300 SEK and ends with 5600 SEK for year 20, which is the maximum number of years a patent can be valid [6].

Priority is a very important concept when dealing with patents; it gives the inventor a possibility to apply for a patent in other countries within 12 month with the original application date set in new application. A short example which shows the importance of priority:

   2. A competitor publishes an article about the invention in June 2010.

Since an application was made a year earlier the application date will be treated by USPTO the same way as if the application was made in January 1st 2010. The published material under these 12 month which otherwise would invalidate a patent application, will be disregarded by the USPTO [6].

Conflicts regarding patents happen all the time. It usually involves the dispute on who are allowed to use the result of a certain invention or if patent should have been approved in the first place. The US differs from the rest of world in the sense that in US the person who first invented something has the right to use the invention regardless of whether he applied for a patent or not. This has caused a lot of conflicts and therefore many countries have put pressure on the US to change their legislation [5].

Another example of a common conflict is who owns the right for a new invention, the company where the employee works or the employee himself? Depending on the country where the patent application is filed, the answer to this question might vary. In Sweden, if the invention is developed within in the same field as your work assignments, the patent normally belongs to the company.
By owning a patent it does not mean you have to manufacture a product that uses the patent. A patent is often licensed to customers or is used to block competitors so that they cannot produce products which infringe on the patent [5].

2.1.3 Trademark
Non-technical aspects can also be of great importance when protecting the commercial interest of a product. The most common are trademark and design protection [5]. A good trademark is a characteristic used to differentiate and promote a product or service from others. Having a good trademark can be make the product or service to stand out from the competition even if the technical merits are similar [7]. The trademark can consist of a name, a symbol, a tune or a sign [5].

Examples of trademarks are:

- Combinations of letters and digits e.g. BBC, ICA, Levi-501s
- Personal names: Tiger Woods, Carolina Klüt
- Jingles e.g. Intel inside tune, Hemglass tune
- Colors e.g. Red Bull (blue/silver/red), Löfbergs lila (the purple color on the coffee package)

A registered trademark is usually in force for 10 years and the period can be extended indefinitely in 10 years increments as long as a fee is paid. The same as for patents, a trademark is national and only valid in the country where the application was filed. There are also similar agreements between countries, making it possible to register a trademark in multiple countries in one filing. The most important is CTM, Community Trademarks, which is a system within the EU for unified trademark registration. One other known association is ARIPO (African Regional Intellectual Property Organization) [5].

To register a trademark is significantly cheaper than to register a patent. As an example a CTM-registration would cost about 20,000 SEK with a fee of about 25,000 SEK every 10 years to renew. A patent in the same area would cost much more [5].

2.1.4 Design
Design is something related to the appearance of a product. When it comes to design protection it is the question of a pure shape or appearance for a product [8]. Design evolved from copyright law to help artists protect work such as industrial or handicraft from reproduction. The laws for design production are not related to the artistic design of the product; instead it focuses on how it is shaped, the characteristics of the lines, contours, surface structure etc., and the materials used when manufacturing the product [5]. Design protection can be applicable for wide range of products e.g. patterns of car tires or cell phones [8].

Design protection as with patents and trademarks are national and only valid in the countries where they are approved. In the same way as with patents and trademarks, there are international agreements for making it easier to register designs multiple countries. Designs can usually be renewed up to 25 years [5].

It is possible to have both trademark and design protection of the same product. E.g. Coca-Cola Company had a design and trademark protection on their Coke bottle from 1915. The design protection has since long expired, but the trademark protection is still active [5].
2.2 Terminology of Ipendo Platform
This section summarizes some basic terminology in Ipendo Platform relevant to this master thesis project. For this project the most important part to understand is the document handling capabilities of the product and the object types able to store documents. The source for this section was both Ipendo Platform Online Manual and my own experiences of working with the product and the source code.

2.2.1 Nodes
All objects in Ipendo Platform are represented as nodes in a hierarchical portfolio tree in the database. These nodes can be of different types e.g. cases, matters, families. Business logic dictates how data can be organized. For example matters can be stored on cases or families but not the other way around. Below are explanations of the different object/node types.

2.2.2 Cases and Families
In the platform customers usually divides their portfolio in to different families where a family represent an invention or other intellectual property. The family is then divided into different cases where a case for example represents the patent application in a specific country or region.

For patents a family usually contains all cases originating from same filing event (first filing) from where priority is claimed. When it comes to trademarks, design and domain names; the parent family normally contains all rights covering the same trademark, design or domain name. Even if this is the most common way of organizing cases it is not enforced by the platform and can be based on any principle. The only limitation is that cases can only be stored in a family with the same IP type.

2.2.3 Matters
Matters are objects for storing information. The information stored varies between different types of matters but all matters can be associated with documents, actions and costs. The customized sets of information can for example be “Agreements”, “Licenses”, and “Conflicts etc.

Matters must be associated with a parent in the portfolio tree. Valid parents are cases, families or a dedicated matter node. Cases and families aggregate and show information about matters associated with the case or family. A matter can at any time be moved to a different position in the portfolio tree. If for example a dedicated node is used to store all new inventions matters, then an invention matter can be moved to a related family or case once a patent application has been filed.

2.2.4 Events
Events are used for registering different occurrences over time in a case. They can then be used to monitor all activities during the process of a case and all related data will be arranged under each particular event.

An important property of events is that they can trigger “Country Law Rules” or “Business Rules” and create actions and reminders, if the rule engine module is activated. Country Laws Rules is rules based on jurisdiction in the country or region where the IP application is being processed (e.g. patent application due dates) and business rules are rules based on internal company policies.
2.2.5  Documents
Ipendo Platform supports uploading and downloading of documents on matters, events, cases and families. Each document has a document type, a title and a document responsible. The time and date of the upload are also logged when uploading.

Uploaded documents can be found under the “Documents” tab in each matter, case or family.

They can also be found in the top main menu under Items -> View Items -> Documents.

Here documents are shown as rows in a list view. From the toolbar on the top the user can search, edit setting and export documents. Each reach row can be filtered to narrow the view of documents displayed.

2.3  Uploading a document to a case
This section describes the main flow of a common use case; uploading a document (received as an attachment in Outlook) to a case in Ipendo Platform. In order to understand the problem solved by this project it is important to look at the work flow in the existing system. See the current workflow in the activity diagram below:
The user must repeat this workflow for each mail he wants to upload files from. Especially having to switch application and keeping track of where on the local computer the files are stored is a hassle for the user.

See full screenshots of the process in “Appendix B – Uploading a document from Platform interface”
2.4  Ipendo Platform Design & Architecture

This section first gives a short introduction to the principle of multi-layer architecture and the proceeds with describing the state of the current architecture and implementation in Ipendo Platform.

2.4.1  Multi-layer architecture

Having an implementation that contains a mixture of user interface, business logic and database access in the same classes can cause a lot of problems in a product. Such products can be hard to maintain, because interdependencies between components causes strong ripple effects when changes are made anywhere. Also high coupling with strong dependencies between classes makes them difficult or impossible to reuse. When adding new user interfaces it often requires cutting and pasting business logic code which then has to be maintained at multiple places. The same applies for data access code being cut and pasted among business logic methods [9].

One very common way to address this problem in enterprise applications is to have three independent layers:

1. The presentation layer presents a graphical interface with a login screen to the user. The user enters a username and a password.
2. A function call is made from the presentation layer to the business layer to verify the user.
3. The business layer calls the data layer to retrieve the password for the user.
4. A comparison is made in the business layer between the entered password and the password from the database. The result is sent back to the presentation layer.
5. The presentation layer informs the user via a GUI if the login was successful or not.

Even though the concept of layers and tiers are often used interchangeably, there are some distinct
differences. A layer is a logical structure in a software solution residing on the same machine; a tier represents a physical structure in a software infrastructure [10].

2.4.2 Current state and architectural view of Ipendo Platform
Ipendo Platform started out as a small one-man project (a master thesis actually) and has evolved into a quite large project with about 170,000 lines of code, a development team of 15 persons and a development budget of about 10 million SEK (2009). It has become quite obvious to me after working with the product for a while and talking to other developers at Ipendo Systems that the architecture was not very well planned at the beginning (instead focus was to get a product into the market early) and the documentation for the older parts of the product are scarce or non-existing. This proved to be a challenge when understanding the current implementation, designing the architecture for my project and reusing functionality in the existing product.

Note that some of the problems with Ipendo Platforms described here have been resolved between I started on this master thesis project and completed the report. Ipendo Systems also wants to point out that development of new modules is much better planned and designed. Also this is mainly a problem with maintainability and re-usability of the current code base and not an issue regarding security. The security of Ipendo Platform has been verified several times both internally and by an external company (specializing in IT security verification).

Lately Ipendo Systems has put in some effort to enforce what they call a “Multi-tier architecture” in the product [11]. However since the layers are mostly accessed as precompiled assemblies, the architecture more fits the description of the multilayer architecture described above and not a multi-tier architecture. A simplified static view of the platform architecture is shown below:
All new development is supposed to conform to this structure and old code is gradually re-written as well.

As previously hinted the illustrated architecture view above is more of a goal than the actual reality of the current implementation. Here are some examples to of the problem with the current solution:

1. Lot of the old code has no layer separation. E.g. part of the user authentication is performed in the presentation layer

2. Some parts of the product (especially the old parts) lack technical design and architecture documentation which makes it difficult to maintain and understand.

3. Ipendo Platform has a “database driven design” which means that a lot of what normally would be called business logic is located in stored procedures in the data base. This approach gives some performance and practical advantages (e.g. patching part of the product without downtime) but is also one of the causes for making the implementation hard to understand and maintain.
2.4.3 Database configuration

Ipendo Platform has one main configuration database (the Live one is located in UK). Its main purpose is to store information about users, companies and database configurations and act as the first entry point at login. It stores user credentials, to which databases the user has access, which database server the database is stored on and which modules that are activated on each database.

The customer databases contain all the data about the customer’s IP portfolio, user credentials (kept in sync with configuration database) and access definitions for the users with access and application configurations. Depending on the region, the database can be located on different servers. The geographical regions are US and Europe but there are also multiple servers per region. If the database is located in the US, the user will also be re-directed to a web server located in the US.

2.5 Security

Users are authenticated to Ipendo Platform by login in from a web browser with a username and a password. The username is the same as the main e-mail address registered in the customer database. Communication between the web browser and the web server is done by secure HTTPS connections.

The portfolio security model is quite complicated. It uses a combination of user classes and access definitions to enforce security. Most of these features can be configured directly from the user interface.

Second layer authentication

Some customers have requested as an extra security precaution to have a second layer of authentication. After a successful login a password is sent to the main e-mail address registered by the user and a second login dialog is presented in the web browser. To complete the authentication the one-time password must entered in the second layer password dialog before the password expires. The password expiration can be configured per customer database.

2.5.1 Technology used

Ipendo Platform is solely based on Microsoft .NET ASP Framework 3.5 and runs on IIS 7.0 (Internet Information Services, web server software) on a Windows 2008 server platform. Data storage is
implemented by the use of databases and is currently running on Microsoft SQL 2008. All customers have different databases as an extra precaution to avoid leakage of information.
3 Service Oriented Architecture

The design philosophy I decided to use for the server side of my solution was Service Oriented Architecture. Therefore, this chapter explains the basic concepts of service oriented architecture for the reader not previously familiar with the subject.

The traditional way of automating business tasks has been to identify the business requirements and then building corresponding solution logic for the specific process. Because the applications were tailored to meet a specific need, the ability to gain further value from them was limited. When new requirements and processes were introduced it forced significant changes or a complete re-write of the application. The creation of new solution logic for each application and process could many times lead to redundant functionality and each new or extended application adds to the bulk of the IT environment inventory. That in return leads to increased hosting, maintenance and administration demands which can inflate the need of resources for the IT department until it becomes a problem for the whole organization. Applications built only for automation of specific business processes in mind are generally not designed for interoperability with other processes. If the need to exchange data between these types of applications occurs later on it can result in a jungle of convoluted integration architectures or introduction of large middleware layers [12].

The experience of several generations of traditionally distributed solutions and an amplified severity of the above described problem led to the creation of service-orientation. It represented an evolutionary step that combined successful design elements of the past with new design elements that leverage conceptual and technology innovation [12].

Service Oriented Architecture (SOA) has been a buzzword in the computer industry in the last couple of years. Even though the definitions vary most experts agree that “SOA enables discrete business functions from various sources to be modularized and distributed as services” [13].

OASIS (Organization for the Advancement of Structured Information Standards) defines SOA as the following [14]:

“A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.”

One of the goals of SOA is to promote loose coupling between software components so that they can be reused and consumed by clients in different applications or business processes. Services are software components with well-defined interfaces that are independent from implementation. The
service interface for the services is separated from the implementation since clients who consume the service should not have to be concerned with how those service requests are executed [15].

As always when it comes to enterprise IT, cost savings are an important argument when implementing a new strategy; this of course applies to SOA as well. It is argued that Service-Oriented Development of applications (SODA) is estimated to reduce the total IT expenses the in long term compared to traditional development. The savings also are exponential since when the library of services created for the business increases, so do the opportunities of service reuse [16].

3.1 Service Design paradigm
A design paradigm is an approach for designing solution logic. When it comes to services there are some important principles for designing services that are truly service-oriented [12]:

3.1.1 Standardized Service Contract
Standardized Service contract represents maybe the most fundamental part of service-orientation because it requires special considerations to be taken into account when designing the public technical interface of a service and assessing the nature and quantity of content to be published as part of the service official contract.

Here a lot of emphasis must be put into specific details of contract design like the manner in which services express functionality, how to define data models and data types, and how policies are asserted and attached. The constant focus is to ensure that services are optimized, appropriately granular, and standardized to make sure that endpoints established by services are consistent, reliable and governable.

3.1.2 Service Loose Coupling
Coupling is the relationship or connection between two things. The level of coupling is equivalent to the level of dependency between service contracts, implementations, and service consumers.

The principle of “Service Loose Coupling” promotes independent design and evolution of a service’s logic and implementation while still ensuring interoperability with consumers relying on the service’s capabilities. Several types of couplings are involved when it comes to services, each which can impact the content and granularity of its contract. Achieving the right level of coupling requires practical considerations to be balanced against service and design preferences.

3.1.3 Service Abstraction
This principle ties into many aspects of service-orientation that on a fundamental level emphasizes the need to hide as much of the underlying details as possible. By doing so it also enables and preserves the previously described loosely coupled relationship. When assessing the appropriate level of abstraction, various forms of metadata come into the picture. The extent of abstraction can affect service contract granularity and also the total cost and effort of governing the service.

3.1.4 Service Reusability
Service orientation strongly advocates reuse; therefore it has become a core part of typical service analysis and design process. The introduction of mature, non-proprietary service technology has made it possible to maximize the reuse potential of multi-purpose logic in a way that was previously unthinkable.
The principle of Service re-usability focuses on positioning services as enterprise resources with agnostic functional contexts. Many design considerations are raised to ensure that individual service capabilities are appropriately defined in relation to an agnostic service context and that the requirements of reuse have been fulfilled.

3.1.5 Service Autonomy
To carry out services consistently and reliably, the underlying service logic needs to have control over its environment and resources. The principle of Service Autonomy defines the extents to which other design principles can be realized in a production environment. It does that by encouraging design characteristics that increases the service reliability and predictability. This principle raises various issues related to service logic and the actual environment of the service’s implementation. When trying to archive a suitable measure of autonomy, considerations as isolation levels and service normalizations are taken into account; especially for services that are frequently shared and reused.

3.1.6 Service Statelessness
Management of excessive state information can compromise the availability of a service and limit its scalability potential. Services should therefore ideally only remain stateful when required.

3.1.7 Service Discoverability
In order to position a service as IT assets with repeatable return on investment they need to be easily identified and understood when an opportunity for reuse occurs. The design must therefore take into account the “communications quality” and individual capabilities of the service, regardless of whether a discovery mechanism such as service registry is being used.

3.1.8 Service Composability
When the complexity of the service-oriented solutions grows, the same applies to the complexity of the underlying service compositions configurations. One of the fundamentals of service-oriented design computing is the ability to effectively compose services. Complex service compositions put demands of service design that have to be anticipated beforehand to avoid extensive re-fittings efforts. Services are expected to be able to participate in a composition regardless of whether they from the beginning are enlisted in a specific composition or not.

3.2 SOA and Cloud computing
This subsection explains the relation between SOA and another popular concept in the IT industry at the time of the writing of this report, called “Cloud Computing”, and why a web based solution like Indepo Platform fits well within both concepts.

Because of the significant advances in Information and Communications Technology (ICT), many people believe that computing will one day be offered as utility in the same way as for example water and electricity are offered today. This utility would cover all basic level of computing services that is considered essential to meet the everyday needs of the general community. Many computer paradigms have been proposed to realize this vision, of which the latest one is cloud computing. The term “Cloud” denotes an infrastructure from which business and users can access application from anywhere in world on demand. Thus, rather than running the software on individual computers, it is transformed into a service [17]. One well-known example at the time of writing is Google’s “Google Apps” [18].
A service in the cloud can be anything a consumer finds useful and can be effectively delivered to a customer asking for it. The principle of “everything as a service” applies for cloud computing, but more specifically it can be divided into the following three categories [19]:

Infrastructure as a service (IaaS) – This is typically a platform virtualization environment and can be an alternative to hosting your own physical servers. Amazon EC2 (Elastic Compute Cloud) is an example of an IaaS [20].

Platform as a service (PaaS) – The PaaS provider can deliver a platform that can easily be used by developers without the need to install their own OS and middleware. Examples of PaaS are Google App Engine and Windows Azure [21].

Software as a service (SaaS) – This is basically software provided on the internet without the need for the end customer to host their own environment and offered as a subscription instead as buying a single software license. It offers predictable costs for the customer and often stable income for the provider [22]. Most of Ipendo’s customers run Ipendo Platform as SaaS.

Notion of services in cloud computing is much broader than that of SOA but notion of services in SOA fits very well within the notion of services in cloud computing, particularly this is true for SaaS. This will allow SaaS to take advantages of the technology, mature architectural style and processes, and best practices of SOA. Another way of looking at it is that SOA deals with application services, forming a subset of SaaS. Therefore one can say that SOA can be a part of cloud computing [19].

However cloud computing and SOA are not the same thing and are not interchangeable. SOA is an approach to architecture while cloud computing is a way of deploying aspects of an architecture, including SOA [19].
4 Middleware

“In a distributed computing system, middleware is defined as the software layer that lies between the operating system and the applications on each site of the system” [23].

This chapter will describe some of the most common middleware for implementing SOA. All middleware will be described based on how transport, service contracts, and service discoverability are handled.

The standards and technologies described below were all evaluated during the design phase regarding on how well they live up the requirements for this project.

4.1 CORBA

The Common Object Request Broker Architecture (CORBA) is a standard developed by Object Management Group (OMG). OMG is a non-profit consortium of 700+ companies created in 1989 and promotes the theory and practices for development of distributed operating systems [24].

CORBA is one of the most well-known middleware standards for enabling exchange of information, independent of hardware platforms, programming language, and operating system. OMG does not produce software or implementation guidelines so there are multiple CORBA implementations available, both open-source and commercial [24].

The Object Management Architecture (OMA) created by OMG, defines various high-level facilities that are necessary for distributed object-oriented computing. The most important part of the OMA is the Object Request Broker (ORB), a mechanism providing object location transparency, communication and activation [24].

CORBA is today more and more marginalized in favor of web-services. One of the reasons is that it suffers from the same type of weakness as other competing standards; a problem to communicate through internet firewalls. [25].

4.1.1 Transport

When the CORBA-client communicates with a CORBA-service it sends method calls to the ORB. The ORB implements the request to the remote object by locating it on the network, communicating the request from the client, waiting for the result and returning the result to the client [26].

The ORB provides location transparency. The request mechanism used by the client is exactly the same regardless of the objects physical location. The client cannot tell the difference if the object is in the same process, in the same building or on the other side of the planet. These ORBs communicates over the internet with Internet Inter-ORB Protocol (IIOP). IIOP then uses TCP as transport protocol [26].

Common Data Representation (CDR) is a way of mapping OMG IDL to low level data types to be able to use them in communication between components in a network [27].

4.1.2 Service Description

In the CORBA specification it is specified which interfaces and facilities the ORB must provide to be compliant [28]. The language and location independent interfaces are defined in the CORBA Interface Definition Language (IDL) [24].
4.1.3 Service Discovery
In CORBA there are two different standards available for service registration and discovery: Naming Service and Trading Service.

4.1.3.1 Naming service
Naming Service is a standard service for CORBA-applications. The service provides an association between a name and a CORBA-object, making it possible for the client to find the object by name only [29].

4.1.3.2 Trading service
In CORBA trading service the objects are not associated by name. Instead an object is found based on the type of service it provides. A client can easily find all objects associated with a specific service [29].

4.2 Jini
Jini was developed by Sun Microsystems for building service-oriented architectures. It has some similar concepts as CORBA, but it differs in the sense that it is based on Java and deriving many feature purely from this Java basis (such as the use of RMI and Java serialization). Also it is an open architecture in the way that it allows other communication protocols (same as CORBA) [30].

4.2.1 Transport
Jini allows different communication protocols to be used but the most common is to use Java RMI (Remote Method Invocation). Java RMI is a communication technology that is the Java equivalent to RPC (Remote Procedure Call) that by the use of the Java serialization mechanism enables Java objects to be packaged for transport. It allows Java objects to be passed back forth between a server and client and enable compiled code to be moved and executed in different virtual machines [31].

4.2.2 Service Description
A service is defined by its API, declared as a Java-interface. A service definition on how the service is utilizing the network must be included since this is not defined globally [30].

4.2.3 Service Discovery
A service connected to a Jini-network is registered by a published java object which implements the API of the service. The client finds the service by searching for an object supporting the service API. When the client receives the object it downloads the code needed for communicating with the service. There are two steps when a service connects to the Jini-network [30]:

1. Discovery: The component is connected to the network and sends out information about itself. The Jini-network Lookup function listens for such packages, at discovery it returns its own interface to the component.
2. Join: With help from the interface, the component sends information for registration to Lookup. The register information is information of which services the service provides and the code, or pointers to the code, needed for others to utilize the service.

4.3 Web Services
Web services are defined through various industry standards supported by all the major vendors [12]. World Wide Web Consortium (W3C) gives the following definition of a web service [32]:

---

22
A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP (Simple Object Access Protocol) messages, typically conveyed using HTTP(Hypertext Transfer Protocol) with an XML serialization in conjunction with other Web-related standards.

4.3.1 Transport
Web services use HTTP, a protocol describing how messages are formatted and sent over the network. An HTTP session is based on the idea of request-response transactions, where a request is sent and an answer is returned. The communication is usually carried out with TCP/IP over the internet [33].

Since communication is done over port 80 (standard port of HTTP) one of the advantages with web services is that there are less problems with firewalls. For exchanging messages the protocol SOAP is most commonly used but there are also competing technologies like REST (Representational State Transfer).

4.3.1.1 SOAP
SOAP, Simple Object Access Protocol, is a protocol for exchanging structured information in a distributed environment [34]. There are no instructions in the SOAP definition on how to build applications; instead the protocol is solely based on how data and requests are sent between applications. To communicate SOAP is utilizing XML-documents (XML, stands for Extensible Markup Language and was developed to fulfill many different design goals), thus making the protocol platform independent. A SOAP application must do the following when receiving a message:

1. Identify all parts of the message
2. Verify that all the mandatory parts identified in the message are supported by the application and process them. If this is not the case, discard the message.
3. If the application is not the endpoint, then all parts identified in step 1 should be removed before the message is forwarded.

4.3.2 Service Description
The Web Service Definition Language (WSDL) is an XML-format that describes the web service as a set of endpoints. It is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate [35]. Many development tools e.g. Microsoft Visual Studio has support for generating the WSDL-definition based on a class definition written in a high-level language (like C#).

4.3.3 Service Discovery
There are several methods for service discovery and many vendors provide their own solution with various level of dynamism. One example of a standardized method is WS-Discovery which many vendors support [36].

4.3.4 REST
The above definition of a web service is the most common but recently REST which is a much simpler implementation of a web service has become popular. Service consumers use basic HTTP operations like GET, PUT, POST, and DELETE. The request is a simple HTTP operations and the response is in the form of a simple XML text. There are no other non-functional requirements except the basics, e.g.
transport-level security with HTTPS. Compared to frameworks for Web services using WSDL and SOAP there is less functionality for security, availability and transaction management. But REST has become more and more popular for its simplicity and it works well for read-only capabilities [37].

4.3.5 Windows Communication Foundation

Windows Communication Foundation (WCF) is Microsoft framework for building service oriented applications:

“Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data.” [38]

WCF has support for creating both WSDL/SOAP and REST Web Services.

Session handling and Security in WCF for WSDL/SOAP Web services

The WCF sessions are initiated and terminated from the calling application/client and messages are processed in the same order as they are received. A session groups messages into a conversation. There are several alternatives for how this grouping is done, e.g. it can be based on a shared network connection or based on a common tag in a message body. There is no general data store provided for WCF session so the storage of session data must be solved on application level [39].

There are two main security modes used in WCF: Transport security mode and message security mode. Transport security uses a transport-level protocol, e.g. for securing the transfer. The advantage is that it is widely adopted and has good performance. Its disadvantage is that it only secures messages from point to point. Message security supports both security based on WS-security (a standard for implementing secure web services) and other standards and the security is directly applied to the SOAP message. The advantage is that it is independent from the transport protocol and ensures end-to-end security. It is however several times slower than transport security because of the verbose XML nature of the SOAP messages [40]. There is also a security mode called “TransportWithMessageCredential” which uses the transport layer to secure the message and still every message includes rich credentials needed for some other services (e.g. grouping the messages into the same conversation/session). The combination gives the performance advantages of transport security and at the same time provides the richness in credentials of message security (but only for point-to-point security) [41].
5 Design

This chapter describes the goals, discussions, pre-study and the decision which lead to the chosen design.

5.1 Design goals

Security, easy distribution of updates and maintainability, platform independent server interface and re-usability were the main goals when designing the software.

Most data handled by the platform are highly confidential; therefore security is a very important issue for users. For the architecture this implies that all communications and storage of data have to be encrypted and user access to the platform must be properly authorized.

One of the advantages of providing the platform as an online service is that it is easy to distribute bug fixes and software enhancements without user interaction. A desktop application (such as an Outlook add-in) must be installed locally on the customer’s computer and therefore making updates more troublesome. So the solution had to be designed to receive software updates in a way that would ideally never require user interaction or resources from support staff.

The reason to strive for platform independent integration is to have the possibility to extend the integration to e.g. mobile devices or software on other computer platforms.

One other important goal for the design and implementation is the ability to extend and re-use the new components in other products. This means that there should be low coupling between different parts/layers in the solution. E.g. it should be possible to replace the GUI or build support for other clients like Lotus Notes without changing/re-building the way the client communicates with the server.

A very important goal is also that the new solution must be able to integrate and share code base with the current implementation of Ipendo Platform.

5.2 Pre-study & Important design Decisions

This section describes all the important design decisions and the motivation for them. Most of these decisions were reached after the pre-study, which was conducted both as literature study and by creating a number of “proof of concept” prototypes to try out some hypothesis regarding the technical solution. I also had a great deal of help from my supervisor at Ipendo System for many of these decisions.

The first fundamental problem was to enable communication between Microsoft Outlook and Ipendo Platform. The current version of Ipendo Platform does not have any other ways of interacting with the product except a graphical web interface. I found three alternatives how to solve this:

1. Make a client/plugin that interacts or embeds the current platform (web) interface. The advantages would be that no new server side part would need to be developed and maintained and also not risk creating any new security flaws in the product. The disadvantage however is that such a solution would be difficult to implement since the web interface in not well suited for other clients than a web browser operated by a user would and be vulnerable to even small changes in the GUI.
If the web interface is instead embedded and displayed in the client application a lot of vulnerabilities of an “unstable” interface are removed but it would also mean that the user experience would be almost the same as using the web interface and it would therefore be difficult to provide any advantages (e.g. when it comes to file handling) compared to using the web interface directly.

2. Directly connect to the database server/database from the client program when reading/writing data. However exposing the database directly on the Internet would break the current Ipendo security model and was therefore not be an acceptable solution.

3. Create a new server side interface/API that the client can interact with. The advantage would be that it would be easier to create a suitable and stable interface. The disadvantages would be that it could create new security flaws if done incorrectly and it would also be a new part of the product that would need to be maintained in new releases. It must therefore be designed and implemented in a way that enforces the current Ipendo security model, re-uses as much as possible (preferably all) of the current business logic and provides an interface/API not closely tied to a specific platform or software provider.

After some considerations and discussion with my supervisor at Ipendo Systems it was agreed that the third suggestion is the best in this situation. The first idea of embedding the web page in an application would in some situations be an attractive choice (especially since it would be easy to implement and maintain) but since one of the goals was to improve the workflow it is not suitable in this situation since it would not provide any advantage when it comes to handling attachment from Outlook.

In order to provide an interface as stable as possible I decided to use an established middleware and evaluate the quality of the service by the principles of service oriented architecture as described in the previous chapter. Main reason for using an existing SOA middleware was to save development time and make it easier to standardize the services (e.g. the data communication format).

I performed literature studies of the standards/technologies CORBA, Jini and web services (SOAP, REST). The conclusion was that a web service would be most suited for the task since it had the following advantages over the other technologies:

- It is standardized and the frameworks for it are still actively developed so compared with competing technologies it seems more future-safe.
- Well supported within the .NET-framework. This was a very important point since it meant that integrating the solution with Ipendo Platform would be much easier. Even though third party solutions exist to use, for example CORBA with .NET, it is not widely supported and documented.
- Since a web service would use http (or https) as network protocol it should work more seamlessly through firewalls than the other alternatives.
- Encryption with SSL could easily be added on the transport protocol (also true for some of the other technologies).

The most recent technology for creating web services (and other service oriented applications) in .Net is Windows Communication Foundation (WCF).
Improving performance for the document upload was not a goal of this project but the performance should at least be a par with the existing solution. That is, the new solution has to be tested so it can handle the same file sizes and numbers of files as when the task is performed from the web interface and make sure the performance is approximately the same. Except sending documents also some other data like user information, database configuration etc. needs to be sent between the server and the client. No other special considerations were taken in the design or architecture regarding performance since it was assumed that using the same transport protocol and similar way of saving the document to the database would also result in similar performance.

**Re-use vs. new architecture**

One of the most difficult design decisions was which parts of the current solution in Ipendo Platform that could be re-used and which parts that should be created from scratch. As mentioned earlier Ipendo Platform is implemented in a three layer/tier structure with a Presentation Layer (web), Business layer and a Database layer (see architectural overview in chapter two). The database layer then interacts directly with the database server.

Since one of the fundamental ideas of a multi-tier architecture is that the layers should be interchangeable it should theoretically be an easy task to add an additional presentation layer in form of a server interface (e.g. a web service) which interacts with the business layer. As described in chapter two the problem with Ipendo Platform is that only new parts of the platform are written in the new structure but still a lot of the old parts are not as strictly separated. For example part of the login procedure is still handled in the Presentation Layer. Another fundamental flaw discovered during the pre-study is that the http session is frequently used by all layers in the product (even the data layer). This session storage can only be used by an http session and not by a WCF web service and made re-using existing functionality difficult. The alternatives to solve this problem were:

1. **Re-write the session handling in Ipendo Platform to separate the technology dependent session type and the data held by it.** In other words create a generalized session handling that can be used by different types of presentation layers. It would also require moving all needed business logic found in the presentation layer to the business layer so that it can be re-used by my web service. Since session data are used in almost all parts of the Ipendo Platform it would probably be a quite tedious and time consuming task and would require lengthy regression tests of the platform. The advantage would be that existing logic could be re-used and the risk for introducing new problem (e.g. with security) is reduced.

2. **Re-create all basic functionality (like login and session etc.) in a new architecture and use this for exposing the service interface.** It would mean that some of the functionality would be duplicated which would make the solution harder to maintain. There is also a risk that new security flaws could be introduced so this must be done with caution.

After some considerations and discussions with my supervisor we came to the following conclusions:

- Re-building the product during the course of the master thesis work is too time consuming and there would not be any time left for building the new solution.
In the short term it would therefore be best with alternative 2. The new architecture must however be built in a way that makes it possible to merge all the duplicated functionality later and all changes to the product must be documented.

Datasets or Object-relational mapping

All old parts of Ipendo Platform returns the database queries to ADO.NET (ActiveX Data Objects for .NET) Datasets that are sent from the data layer to the business layer where it is processed. The advantages with datasets are that they are straightforward and easy to use (basically it is just a table from the result set of the database query) and it is also easily translatable to and from XML. The drawbacks are that datasets is not as conceptually suitable for object-oriented applications and can lead to “spaghetti-code”, database driven design (too much logic in database layer instead of business layer) and result in applications that are difficult to maintain. I wanted to create a cleaner domain driven design where business objects such as “users” and “documents” could be used. There are two ways of solving this:

1. Use an ORM framework (Object-relational mapping). An ORM converts data from incompatible type systems in relation databases and object oriented programming languages. Thus in practice creating a “virtual object database” so that objects instead of datasets can be used with an object oriented language [42]. There are both free and commercial solutions available. Microsoft’s ORM solution is called ”ADO.NET Entity Framework” and is included since .NET 3.5 SP1.
2. Create my own business object structure and a data transformation layer that transform the data sets into business objects/entities before they are sent to the business layers. In other words, create my own simple ORM.

I choose alternative 2 because of the following reasons:

- ADO.NET Entity framework is a very complex framework that would take some time to get familiar with and it is not obvious that the extra complexity would add any advantage to the solution. Using ADO.NET Entity Framework might also require changes in the data model and that would break many parts in Ipendo Platform and make re-use of stored procedures difficult.
- The first version ADO.Net Entity was not well received by the developer community and had many known issues [43]. There were no obvious other candidate either.

The chosen approach does not actually put the domain related operations in the business objects as recommended according to domain driven design and general OO-principles but are instead handled by the business layer. The arguments for still implementing this “broken” domain driven design was the following:

1. It would be harder to merge the new solution with Ipendo Platform if the separation of business logic were not compatible with the current solution.
2. The implemented solution is still better than using and sending datasets between the client and the server since fewer method calls will be needed and the structure of the entities is still easier to understand and use.

Security

Creating a secure solution has many aspects, from choosing technology and configuration to design and implementation. There were also some functional requirements (e.g. second layer authentication) that needed to be fulfilled. Before I started designing the security model of the solution I had already come to some decisions about the technical solution. The server side would be a WCF web service and a the session data must of course be accessible for the WCF service but the session context data should be stored in such a way that is not technologically dependent on WCF.

As mentioned earlier there are two basic types of transfer security included in WCF, transport protocol encryption and message encryption. The advantage of message type security which in this case would mean encryption of the XML SOAP messages is that it is transport protocol independent, more easily extendable and provides end-to end security (instead of only point-to-point). Its major drawback is that the performance is much worse than using only a transport security mode. WCF supports several kinds of message encryptions. Most of them are branch-standards and would fulfill the goal of keeping the service interface platform and middle-ware independent. However, even though I tested message security during the pre-study phase, my decision here was to not use message security and instead rely on transport security with SSL for transfer security. The reasons for the decision are the following:

- Only using SSL is simpler to implement and deploy since it would mean that the web service and the web interface in Ipendo Platform would use the same protocols and data ports and therefore eliminate any potential problems with firewalls.
- Having equivalent performance to the previous solution was a more important goal than any of the other advantages gained by using message security.

As mentioned earlier, re-creating some of the basic Ipendo Platform functionality in my own solution has the potential of introducing new bugs and security flaws.

Session handling

As previously concluded a new session handling needs to be built since WCF is not compatible with .NET http sessions. One difference between WCF sessions and http sessions is that there is no general data store associated with WCF sessions. A requirement for the new session handling is that it also must be possible to re-use with other technologies than WCF and make it possible to use a single business and data layer for the whole product. There are some alternatives for solving this:

1. Create a session class that stores all needed information. This class will be instantiated when the session is initiated and store the needed information in the session object for the duration of the session. Using this class with for example an http session, it would simply just store this object in the http session data store and retrieve when it needs to pass it on to the business layer. Advantage of this solution is that is straightforward and easy to implement. It
is still process dependent though and will not work well in an environment of clustered web-servers (same limitations as with http sessions).

2. Store the session data in the database. In practice this would mean that the service itself would be stateless and all the logic of the session is handled by the application. This solution would be much more complicated to implement since it would require that the application keeps track of the sessions (or “conversations”) without help from the framework. It could be done by creating a unique session key that is sent to the client when the session is created and used during the rest of the session to authenticate the client (similar to how http session works). The most obvious advantage with this solution is that it would not be limited to a single process thread and therefore also work in e.g. clustered environment (since the session could be shared between different physical servers). It would also be theoretically possible to share session/state information between different technologies and client computers/devices. The most obvious disadvantage is that it would be more complex solution to design and implement and the responsibility of securing the sessions falls on the server application alone.

Even if the ability to function in a clustered environment or sharing session/state information between the web and external applications would be nice features it is not something that are required of the product (or this project) at the moment. I therefore decided on only developing the alternative 1 solution during the course of this project. The development could also be done in an evolutionary manner; first developing generalized session logic, then later add a backend for storing and retrieving it from the database instead of from memory.

One more thing to mention about the session handling is that the timeout for both the session and the period of validity for second level authentication is something that is configurable in each Ipendo platform customer database. I decided to keep this logic in the application instead of trying to dynamically load this into the web service configuration since it would be less dependent on a specific technology. This means that there is both a time-limit for the actual WCF session and an application based time out. Only the latter one will affect the user as long as the WCF timeout is set higher than the one in the application.

**Client – Outlook plugin, Update functionality**

Most of the design decisions so far have focused on the general architecture and communication between the server and the client. However one of the tasks of this project was also to implement a client that could be accessed from Outlook to allow easy transfer of documents from the user’s inbox to Ipendo Platform. For the client application it was an important to create something that was easy to maintain and could be re-used for other client implementations. It was also important that the new version of the client could be distributed to the user with minimal user interaction.

Since I had no previous experience of creating Outlook (or other MS Office) plugins I first studied some online documentations and tutorials on creating Microsoft plugins and did some prototypes to try to what extent this goal could be archived. What I found out was:
1. There is good support for creating Outlook plugins in Visual Studio and it was possible to add support for web services within the plug-in.

2. It is possible to instead of including everything in the plugin to call an external library using Microsoft’s reflection technology [44]. There are some limitations to this approach since some initialization is done automatically by the framework and cannot be done in the external assembly. One example of this is the ribbon interfaces in Outlook which I found no way to load dynamically. It was also possible to connect to a web service from within the assembly but the implementation would be more complicated and required sending and reading the configuration for the web service client separately.

Embedding as much functionality as possible in an external assembly would make it easy to create a transparent update procedure for most of the functionality since the external assembly could be updated (when starting Outlook) before it is loaded. However, including everything in the plug-in is easier to implement and has less risk for encountering limitations during implementation.

In conclusion, the alternatives I considered for the update functionality was:

1. A stand-alone update program. This program would have to be run separately from Outlook (when it is closed). The advantage would be that all parts of the client could be updated (even the actual plugin part) and would work regardless of how the component separation is implemented. Disadvantage is that it will either require the user to start the update manually or run it in background/system tray.

2. Embed everything but the initialization and the update functionality in an external assembly. At startup the plugin would check for the most recent version of the assembly and download it if a newer version is available. Advantage is that it requires no user interaction as long as none of the “core-functionality” needs to be updated. The disadvantages are that a bug in the “core-functionality” could be troublesome to fix and distribute and that this is an unorthodox way of implementing an Outlook plug-in (thus the risk for encountering more limitations is increased). Also for new (and incompatible) versions of Outlook the update package of the plugin would have to be distributed manually to the customer.

After some considerations and discussions with my supervisor and the Ipendo Support Coordinator I decided on alternative 2, because that solution would most likely be easier to support and maintain than having a separate application to handle the software updates.

**Graphical Interface**

The primary goal for the user interface was that it would provide an improved workflow when uploading attachments from Outlook compared to working directly in the platform.

The GUI would have to consist of two parts:

- Outlook specific part - Menus and menu items to make it accessible from Outlook.
- Windows specific part – Standard windows GUI components
For the Outlook specific part there was not much of a choice what framework to use, only where in Outlook to plug-in the new functionality. To give the user a choice of how to use the plug-in, I decided to add menus in the main window, in the menu for each opened mail and in a context menu for the inbox (right-click pop-up menu). When a menu item is accessed it will call a standard window dialog component.

For Windows there are two standard toolkits (both from Microsoft) that can be used with .NET:

Windows Forms – This toolkit has been part of .NET since the first versions and provides access to native Microsoft Windows interface elements by wrapping the extant Windows API in managed code.

Windows Presentation Foundation (WPF) – Is a graphical subsystem for rendering user interfaces in Windows-based applications and was first released for .NET Framework 3.0. Instead of relying on the older Graphical Device Interface (GDI), WPF utilizes DirectX. One of goals with the WPF is to provide a consistent model for building applications and to separate business logic from the user interface. It has some similarities of with XML-oriented object models such as XUL (XML User Interface Language, developed by the Mozilla foundation) and SVG (Scalable Vector Graphics). One of its strengths compared to Windows Forms is that it provides ways to easily include embedded media and creation of themeable applications [45].

After trying out both Windows Forms and WPF I found Windows Forms much easier to understand and closer to my previous experience with other graphical toolkits. I therefore chose to write my GUI in Windows Forms. No doubt that WPF provided some more possibilities but the learning curve to set up something working was higher and I had no real need for the extra functionality.

5.3 Development tools and technology

For the development of the server service and the client desktop application, Visual Studio 2008, the .NET Framework and the language C# was selected. The main motivation for this is that it is the same tools and technology used for Ipendo Platform thus making integration and re-use of existing functionality easier. The drawback was that I had no previous experience of neither the language nor the development tools before starting this project and that led to a higher learning curve.

Microsoft .NET Framework includes a large class library with lot of pre-built code for common low-level programming tasks. The source code is not compiled directly to machine specific code; instead it is compiled to intermediary level. This intermediary level is called “Common Intermediate Language” (CIL) and the same for all the high-level language bindings. This CIL code is executed in the “Common Language Runtime” (CLR) runtime. The CLR provides an application virtual machine so that the programmer does not need to consider the underlying capabilities of the specific CPU that will execute the program.

The version of .NET framework version used in both Ipendo Platform and my project is 3.5.

5.4 Architectural representation

The architecture is described from different views to highlight different perspectives of the architecture:

- Logical view – A static view over the architecture
• Use case view – Describes the use-cases realized by the architecture

5.5 Logical view

This is the static architectural view of Ipendo Platform. The green boxes represent new components developed during this project. The view of the current architecture for Ipendo Platform has been slightly simplified and modules/components non-relevant for this project have been left out. Short description of the components in the different layers:

**Ipendo platform:**

• Presentation layer (WWW) – Responsible for user interaction via a web-interface.
• Business layer – Implements the business logic for the platform.
• Data layer – Communicates with the database.
Extensions (to Ipendo platform) developed during this project:

- Ipendo Platform Desktop Service (IPDS) – A web service functioning as the communications interface for the client application.
- Entity – Contains the data entity classes. Entities are data carriers for communication between the server and the client.
- IPDS Business – Business logic needed for the project. Duplicates some of the logic from the platform and some of this should later be merged with the business layer of the platform.
- IPDS Data Layer – Accesses data from the database via Ipendo Platform’s data layer or by directly executing stored procedures in the database. Repackages all retrieved data into entity objects before returning them to the business layer.

Client side:

- Ipendo Platform Outlook Plugin (IPOP) – This is the client application and is divided into two parts: IPOP_STARTER and IPOP. The first one implements the Outlook specific properties and acts as a wrapper for the actual plug-in.
- GUI – A graphical interface which takes care of all interaction with user. Is accessible from Outlook main window, context menu and a menu in an active mail.
- Communication layer – Responsible for communicating with the web-service, transfer data etc.
- Utility – Various utility classes for handling e.g. Outlook and icons.
- Entity (proxy) – This is an auto generated proxy class and contains the interface for the IPDS entity layer.

5.6 Use-Case view

The following use-cases have been realized in the architecture.

UC1 Login

UC2 Upload attachment

UC3 Logout

5.7 Use-Case realizations

The two use-cases illustrated with sequence diagram below are “Login” and “Upload Attachment”. It demonstrates the core-functionality of the IPDS/IPOP software and the interaction with the Ipendo Platform.
5.7.1 Login

1. The user signs in using username and password. If IPDS successfully verifies the user’s identity against the platform it will send back all the databases available to the user. The user will be notified if the login was successful or not and presented a list where he can choose one database.

2. IPOP will request to create a session and send back the user object together with the selected database object. If the user can be validated again a session is created and a connection string to the selected database is saved in the session.
This use-case has three main steps:

1. The user selects which attachment(s) to upload in the first dialog and can either choose to browse or search the portfolio (accessed by two different buttons).

2. If browsing: IPDS retrieves all root nodes accessible to the user. It will also retrieve node information such as “display name” and “nodetype” and store it in nodes (an entity class). The data returned will be a collection of tree nodes. IPOP receives the data and displays it to the user in form of an expandable/collapsible tree-structure. When the user expands a tree
objects with sublevels, it will retrieve the objects in the first sublevel in the same way it did with the root-level.

3. If searching: The returned data will be the nodes matching the keyword and the option (“All”, “Family”, “Case”, and “Matter”). The nodes will be displayed as a selectable list. If a tree node is not a valid document save point it will be grey-marked and not selectable. IPOP sends the files(s) to IPDS and IPDS tries to save them to the chosen location. The user is notified on the result.
6 Implementation

This section describes how the solution was implemented with one section for the server side and one section for the client. It covers the technical details about the implementation, problems that occurred and needed to be addressed during this phase and also parts that could not be finished because of the timeframe of the project.

The implementation was done in two iterations where the first part focused on the outlook integration, the graphical interface and the communication with the web service. The actual integration with Ipendo Platform was left for iteration number two since it was not sure I would have time to complete that part.

During implementation I documented where functionality (and sometimes code) were duplicated to make it easier to merge the new branch back to the main branch.

6.1 Ipendo Platform Desktop Service

The illustration below shows the component separation and dependencies implemented in IPDS.

![Component separation diagram]

Many of the service requests will include transfer of data entities defined in the entity package (such as user, document, tree-nodes etc.). The web service receives the data and forwards the request to the IPDS business layer which calls the data layer to retrieve any needed information. The data layer receives datasets from the database and repackages it into data entities before returning the data to the business layer. All needed business logic is then carried out in the business layer and the result is returned to the client through the web-service. Below the implementation of the business entities and the different layers are described.

6.1.1 Business Entities

The Business Entity package contains the classes used as data carriers between the layers. See below part of the Node-class (one of the most used entities):

```csharp
[DataContract]

public class Node
{
    private string displayName;
    private int nodeID;
}`
```csharp
private int nodeTypeID;
private string nodeTypeName;
private int parentID;
private bool hasChild;
private string image;
private string ipType;
private int ipTypeID;
```

Only the properties necessary for the application was implemented in order to keep the business objects (BO) as lightweight as possible (so that no unnecessary data is sent between the server and the client). See the implemented classes for business entities and their properties below:

<table>
<thead>
<tr>
<th>Entity Name</th>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>databaseName (string), companyID (int), secondLevelAuthentication (bool)</td>
<td>Customer DB information</td>
</tr>
<tr>
<td>Document</td>
<td>filename (string), fileContent (bytearray) documentType (DocumentType Entity), documentTitle (string), parent (Node Entity)</td>
<td>Document file and meta data</td>
</tr>
<tr>
<td>DocumentType</td>
<td>displayName (string), documentTypeID (int)</td>
<td>Display name and Document type ID for a configured document type</td>
</tr>
<tr>
<td>Node</td>
<td>displayName (string), nodeID (int), nodeTypeID (int), nodeTypeName (string), parentID (int), hasChild (bool), image (string), ipType (string), ipTypeID (int)</td>
<td>A node which most base objects in Ipendo Platform can be represented as</td>
</tr>
<tr>
<td>Session</td>
<td>validUser (bool), isAuthenticated (bool sessionTime (DateTime), sessionTimeOutLimit (TimeSpan), currentDatabase (DataBase Entity), connectionString (string), securityLevel (int), User (User Entity)</td>
<td>Used to store the information of the current session.</td>
</tr>
<tr>
<td>User</td>
<td>userID (int), securityLevel (int), userTypeID, username (string), password (string), databases (Array of Database Entities)</td>
<td>Used to store user information and the databases available to the current user</td>
</tr>
</tbody>
</table>
6.1.2 Layers

**Service Interface (Presentation Layer)**

The service interface defines which methods to expose to the client and some setting for the individual methods. It was implemented to be as straight-forward as possible and does two things:

1. Keep track of the session data and session logic.
2. Forward the request from the client to the business layer and return the result to the client.

I used three different settings for the externally exposed methods/services:

- IsInitiating – Calling a method with this setting will initiate a new session.
- IsTerminating – The session will terminate after the method call
- IsOneWay – The method will not return any data (and the client will not expect it).

Also the general setting for the services was that an active session was required when calling all service API. It was defined in the header of the service contract as “[ServiceContract (SessionMode=SessionMode.Required)]”. By implementing it this way it made it easy to control the workflow and the security of the services. In order to make the service session aware the WCF settings was changed to “TransportWithMessageCredential”

A self-signed SSL certificate was used during development and imported on the client.

The actual service contract in WSDL format was not written manually but generated by the framework at compilation. To generate the client part of the server interface a Microsoft command line-tool called svcutil was used.

Svcutil generates the client proxy code based on the Web Services Description Language (WSDL) or policy file received from the service. The output is the DesktopService-class and configuration file that can be used by a client application when calling the web service. The DesktopService-class on the client side is as mentioned only a proxy class which means it only contains the structure for available methods etc., while the actual implementation is kept only on the server side. Each time the server contract changed a new client configuration needed to be generated.

Full description of the Service Interface API can be found in “Appendix A – Service Interface API”.

**6.1.2.1 Business Layer**

Ipendo Platform has mostly a very lightweight business layer because so much logic is performed in the database layer but for the implementation of the business layer for IPDS I tried to keep a stricter separation between data processing and business logic. The most difficult part of implementing the business layer was to make sure security could not be compromised.

The implementation of the security for the business layer can be summarized as follows:

1. Compare the user credentials sent by the client with the user credentials in Ipendo Platform at login and initiate the session. At login the password is sent in clear text by the client (but encrypted by the transport protocol), the business layer encrypts it and compares it with the
directly functionality in Ipendo Platform either by forwarding the request to the existing database layer or by directly calling stored procedures in the database.

encrypted password retrieved from the Platform. If valid, the user entity is returned to the client but since the encryption method used by Ipendo Platform is not a one-way encryption (can be decrypted with same key) we risk exposing the encryption key if we send the encrypted password back. Therefore the password is removed from the user entity before it is returned to the client.

2. If the user is validated but the customer database has second level authentication enabled the session will be initiated but the user will not be authenticated. A one-time password will then be generated and sent by mail (to the e-mail address associated with the user) and OTP-settings (like timeout, max number of tries) set in the session. If the correct password is sent within the timeout-limit the user will be authenticated.

3. As an extra safety precaution the client application also identifies itself by sending a name and password before the session initiation. This solution is not 100% since the password is stored in the client binary and could probably be extracted by someone with access to the client software. It will however protect against “black box attacks” to the services and the password can easily be changed if wanted. Instead of storing this password in the web server configuration (which is the default) I instead implemented it by overloading the constructor for the WCF session validator. That way it is possible to load the password from other sources, e.g. the configuration database or configuration file. But for now it is “hardcoded” in the validation method.

4. The session data is only stateful in the presentation layer but the logic is handled in the business layer. During a session when a service is accessed it always first checks if the session is still active by comparing the timeout limit with the current time. If it is still active it updates the session timestamp to the current time. If the time limit has already been met a Time Out exception will be thrown to the client. The session data is of course never sent back to the client since it contains exploitable information (e.g. the database connection string).

5. When data is requested or sent for storage in Ipendo Platform it is necessary to check the access controls against user privileges. Problem with fulfilling this task is that the security model used by Ipendo Platform is quite complicated, there is a lot of special logic depending on for example user class, and almost no documentation for this exists. After discussing this with my supervisor and some of the experienced developers at Ipendo Systems we came to the conclusion that it is best to leave this to later since it is not needed to prove that my solution is working and would be very time consuming to implement and verify for me. Therefore the current implementation only has a placeholder in the business layer for the access control where needed and no finished implementation for this part.

6.1.2.2 Data Layer
This layer is where the actual integration with Ipendo Platform is done and where the datasets are converted into business entities before the result is sent back to the business layer. It accesses functionality in Ipendo Platform either by forwarding the request to the existing database layer or by directly calling stored procedures in the database.
The lack of technical documentation about the existing database layer in Ipendo Platform made the implementation harder than expected and also affected how much business logic that could be moved to the business layer.

6.2 Ipendo Platform Outlook Plugin
This section describes the details for the implementation of the outlook plugin/client applications. The client side was implemented as two projects, one for the Outlook plugin and one for the assembly containing the user interface, communication functionality and various utility classes. See component division below:

6.2.1 User Interface
Creation of the user interface was the difficult part to implement on the client and required several re-designs before it was accepted by Ipendo Systems. Even though I did some simple sketches (on paper) before starting with the implementation it was difficult to predict the look and feel before implementing a working prototype. The first prototype was a wizard-like interface which was easy to follow but required a lot of steps each time an attachment was uploaded and it did not support uploading multiple documents at once. Since the potential users would use this functionality to improve the workflow compared to the web interface, efficiency was more important than oversimplifying the interface. After a couple more prototypes and reviews of the interface by Ipendo System staff I arrived at a solution with the following characteristics:
1. Easily accessible context menus from the inbox

2. The upload attachment dialog gives direct access to both searching and browsing the portfolio tree and the ability to directly setting the metadata properties of the document(s).

3. Support for batch upload of attachments to a single location
See the activity diagram below to for the description of the workflow in the client’s user interface:
6.2.2 Update functionality

At startup the IPOP Starter connects to a web-server and downloads IPOP.DLL to the local user profile before instantiating the main class of the assembly. The following diagram describes the startup sequence:

Screenshots showing the whole main flow can be found in “Appendix C – IPOP”
This ensures that the user always gets the most recent version of the client.

### 6.2.3 Communication

This layer consists of a few classes used for communication with the WCF web service. See table below:

<table>
<thead>
<tr>
<th>Class name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DesktopService</td>
<td>Proxy class used to communicate with the web service</td>
</tr>
<tr>
<td>CustomClientChannel</td>
<td>Used by DesktopService to dynamically create communication channel to a web service with an XML-configuration file as input</td>
</tr>
<tr>
<td>UserSession</td>
<td>Encapsulates client side logic of the session state and communication with the web service so it easily can be accessed from GUI classes</td>
</tr>
</tbody>
</table>

### 6.2.4 Utility

The communication layer and GUI use some common utility classes:

<table>
<thead>
<tr>
<th>Class name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutlookHandler</td>
<td>Handles all Microsoft Outlook specific functionality (e.g. retrieve attachment from mail) that are not GUI related</td>
</tr>
<tr>
<td>ErrorMessages</td>
<td>Used by the communication layer to send error messages to the user. Currently only implemented for WindowsForms framework but can easily be extended</td>
</tr>
<tr>
<td>IconHandler</td>
<td>Encapsulates client side logic of the session state and communication with the web service so it easily can be accessed from GUI classes</td>
</tr>
</tbody>
</table>
7 Testing
The testing were setup to measure the requirements as described in chapter one. All functional and non-functional testing was first performed in test environment (at Ipendo Systems) and then on the live US servers.

7.1 Functional Tests
This part covered testing REQ F1 – REQ F8. Descriptions of the requirements (from the first chapter):

1. Log in to the Ipendo Platform with username and password. It must also handle second level authentication with one-time passwords. No other functionality should be accessible before a successful login. (REQ F1)
2. Show a tree-view of the objects accessible to the user and be able to select a destination to store the document from that view (REQ F2)
3. A search bar in which the user can enter a name or a property to find the object in the account on which the document will be stored. There should also be a search option to filter on “Family”, “Case” and “Matter”. (REQ F3)
4. Upload file(s) to the Ipendo platform after selecting destination from either the tree-view or the search bar result. Valid destinations should be “Families”, “Cases” and “Matters”. The user should be able to specify the document type before it is uploaded and require a confirmation by the user before execution in order to avoid feeding information to the wrong portfolio. (REQ F4)
5. Log out. This should end the session and disconnect the client’s network connection. (REQ F5)

Additional features:

6. Download documents from Ipendo Platform and add as an attachment (REQ F6)
7. Possibility to also upload mail content to Ipendo Platform (REQ F7)
8. Edit properties of a document (after upload) (REQ F8)

The test cases were based on the above requirement specification and the use case descriptions (see appendix). Both main scenario and extensions were tested to make sure all part of the user flow worked as specified.

REQ F1 – REQ F5 were all fulfilled and tested ok but REQ F6 – REQ F8 could not be tested since there was not enough time to finish implementation during the course of the project.

7.2 Non-Functional Tests
The non-functional test regards REQ N1- REQ N4. Descriptions of the requirements (from the first chapter):
1. Platform and application independent server interface. In other words it should be possible in the future to create clients for other applications and platforms, e.g. Lotus Notes and handheld devices. (REQ N1)

2. The transfer of files to the platform must be secure and encrypted. (REQ N2)

3. Updating Outlook plug-in should require minimal user interaction and automatized and not require user interaction except on initial installation. (REQ N3)

4. Performance should be on a par with the previous solution and it should be able to upload attachments up to 30 MB within 1 minute (per attachment) (REQ N4)

Since REQ N1 – REQ N3 were difficult to test by setting up a test case, this is how I verified that they were fulfilled:

REQ N1 – Since the solution kept the outlook specific part separate from the client part that communicated with the web-service, application independency was solved as result of the design of the solution. Regarding platform independence the service interfaces were inspected and compared to SOAP message standard. Since no non-standard data types were used and everything else complied with the web services standards (as described by the standardization organization OASIS). I therefore concluded that the solution is not dependent either on the web service technology or the platform of the client. Another good way to verify this would have been to implement a client using a different technology and platform (e.g. with Java on Linux) but unfortunately I did not have enough time to do that.

REQ N2 – The transfer security is performed with SSL which is an industry standard for transfer security encryption and therefore considered safe enough. The sensitive part (when it comes to security) of the implementation was also reviewed by another developer at Ipendo Systems. There was no time to do any more extensive testing of the security so this is something that needs to be done before the software can be used in production.

REQ N3 – This feature was actually very well tested during development since every change made to the client would result in a new version being downloaded when starting outlook. With the final version of the client I verified this again by changing a minor part of GUI and distributing it to the web server. It worked seamlessly with no need for interaction by the user. I also tested disabling access to the client library to make sure the error-handling worked and that the user was notified when it failed to download the latest version.

The last requirement REQ N4 was tested by comparing the performance with uploading a document of about 30 MB to an account from Outlook to the performance when using the web interface. It passed the performance requirement and performance was about the same as using the web interface.

7.3 Test in a production environment
Running the web service on the US production environment was best way I could find to simulate a real-life scenario with non-ideal conditions like low bandwidth and high latency.
The web service was configured on the live US server with the same settings as on the development environment.

The same test as on the development environment was performed and as expected it was a little bit slower on the live server but all tests were completed successfully.
8 Conclusion and lessons learnt

The project met its goals of improving and shortening (from 11 steps to 7 steps) the user workflow but the most important improvement is that the user will not need to switch application when dealing with mail attachments. Since most of the time spent during this project was to design and implement the infrastructure of the solution there are not that many features available for the user except uploading documents (which was the most important feature requested by Ipendo Systems). However, additional features would be easy to add later because the basic structure is already in place.

See below the objectives (from chapter one) that was set out in the beginning of the project and how they are met with the new solution:

• What would be a suitable technology and architecture for the communication between the Ipendo Platform™ and an Outlook-plug-in?

  **Result:**
  I showed in this thesis that an architecture based on the principles of service orientation which exposes part of Ipendo Platform as a “service layer” is a suitable choice for integration with a desktop application. Since the provided services are implemented in a web service using current industry standard it also ensures re-usability and platform- and technology independence. There are some technological dependencies on the server side (mainly with .NET) but another client can be built (e.g. a Java-plugin for Lotus Notes) without needing to make any changes to the server side.

  When comparing the solution to the “Service Design paradigm” described in chapter 3 it seems that the service layer in most parts conforms to good service-orientation. Some of the qualities like “composability” and “reusability” are something to be considered when adding more services but hard to judge based on the current implementation since not that many services exist yet. Most of the services are stateful and based on whether or not the user is logged in or not. However, no other states exist so this should not affect the availability of the services negatively.

• How should the solution be designed to make it secure and without risk of leaking confidential information?

  **Result:**
  Communication between the server and the client is encrypted with SSL and the implementation of the security critical features (like login) was reviewed by another developer. Unfortunately there was not enough time to do any extensive security testing or formal verification so this is something that remains to do before the software can be used in production. Another thing omitted from the master thesis project in order fit in the timeframe was the portfolio access functionality.

• How should the plug-in be distributed to make installation and software updates seamless for the end-user?
Result:
Initial installation of the software is still necessary but the technical solution developed in this project archives the goal of keeping the client software updated without requiring interaction from the user. As described in the report this is archived by separating the Outlook specific part with the generic client part and keeping the Outlook specific part as small as possible. The generic part which contains everything from the graphical user interface to the communication with the web service is updated to the latest version each time Outlook is started. It does not solve the problem with updating the Outlook specific part but this is something that rarely needs to be done. An improvement to this solution could be to add the functionality of downloading and starting an external installation package and thus making it possible to also updating the Outlook plug-in part.

- What lessons can be learned on integrating Ipendo Platforms with other applications? How could it be made easier?

Result:
The solution obviously works but integration with Ipendo Platform lead to some compromises when it came to the technical solution. It was out of scope of this master thesis but rebuilding part of the product would make it a lot more ready for integration. In the current state a lot of the existing functionality cannot be re-used and must be re-built for each solution. There are ongoing discussions on Ipendo for a “next generation”-version where most of the existing functionality would be re-factored. One goal of such re-factorization could be to make the product more “integration friendly” and have better separation between the different layers.

Some of the experiences and “design patterns” from this solution have been re-used in an external project where integration was not with Ipendo Platform but between Microsoft Office and a SharePoint based document handling system. The experiences and responses from the customer regarding that project have been mostly positive.

I think this master thesis project has reached most of its objectives of designing and implementing a solution to integrate Outlook with Ipendo Platform. The experiences from the difficulties encountered in this project are also something that could be useful for other similar projects and for the general direction of Ipendo Platform.

A lot of the problems encountered during this project were related to the present state of Ipendo Platform. I therefore recommend that before merging my solution or any other similar solution into the current code base the product should be re-built so that business logic can more easily be re-used and accessed externally. Otherwise it is likely that the technical debt in the product will increase and will make it even more difficult to maintain in the future. The best in my opinion would be if the whole or part of the business layer could be re-implemented as a technology independent service layer so that it easily could be exposed both to web services and other SOA middleware.
9 Bibliography


## Appendix A – Service Interface API

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Arguments</th>
<th>Returns</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValidUser</td>
<td>Validates the user against the configuration database. If the user is valid it will read the user- and database configurations (that the user has access to) from configurations database and return it to the client.</td>
<td>Username (string), Password (string)</td>
<td>User entity OR NULL (if user was invalid)</td>
<td>Initiates session = YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Terminates session = YES</td>
</tr>
<tr>
<td>CreateSession</td>
<td>Create a session against a specific customer database. If valid it returns the user entity present in that customer database.</td>
<td>Username (string), Password (string), Database (Database entity)</td>
<td>User entity OR NULL (if user was invalid)</td>
<td>Initiates session = YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Terminates session = NO</td>
</tr>
<tr>
<td>ValidateOneTimePassword</td>
<td>Send a one-time password for the session to authenticate the user. If valid it returns the user entity present in that customer database.</td>
<td>User (User Entity), Password (String)</td>
<td>TRUE/FALSE (bool)</td>
<td>Initiates session = NO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Terminates session = NO</td>
</tr>
<tr>
<td>GetAvailableDocTypes</td>
<td>Get the Documents type configured in the customer.</td>
<td></td>
<td>List of DocTypes (DocType Entity)</td>
<td>Initiates session = NO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Terminates session = NO</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Parameters</td>
<td>Initiates session</td>
<td>Terminates session</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>GetChildNodes</td>
<td>Get all child nodes to for one sublevel of the portfolio tree</td>
<td>parentID (integer)</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>GetMatters</td>
<td>Get the nodes of all matters for a case or family and also the default document storage point</td>
<td>parentID (int), documentMatterNodeID (int out)</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>GetCases</td>
<td>Get the nodes of all cases for a family</td>
<td>familyNodeID</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>GetCaseObjects</td>
<td>Get all objects stored at the case</td>
<td>caseNodeID</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>FindNodes</td>
<td>Get nodes to all cases and/or matters that match the search string</td>
<td>searchString (string), option (string)</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>saveDocument</td>
<td>Saves a document to a node in a customer database</td>
<td>User (User Entity), destinationDatabase (Database Entity), destinationNode (Node Entity), newDocument (Document Entity)</td>
<td>TRUE/FALSE (bool)</td>
<td>NO</td>
</tr>
<tr>
<td>IsAuthenticated</td>
<td>Used by the client to check if the session is still active or have timed-out</td>
<td>TRUE/FALSE (bool)</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>Terminates the session</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------</td>
<td>------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EndSession</td>
<td></td>
<td>One way = YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initiates session = NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Terminates session = YES</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
11 Appendix B – Uploading a document from Platform interface

11.1.1 Save attachments
The first step is to open a mail with attachments in Outlook:

1. Select the attachment to be uploaded
2. Right-click and select “Save As…” (this will open a save dialog)
3. Select the destination (e.g. the Desktop folder)

11.1.2 Login

1. Open a web browser and navigate to the login page for Ipendo Platform

2. Enter your username and password and click on “Login”.

3. Select which client database to use and click on login. This step is only showed if the user has access to several databases.

11.1.3 Find destination

After login the user is brought to the main view:
The user can now either search for the case in the quick search bar or browse the portfolio tree.

11.1.3.1 Portfolio tree

1. Browse the portfolio tree by clicking on “Portfolio” -> “View Portfolio Tree”

2. Locate the patent family in the portfolio tree to which the case belongs.
3. Select a case from the “case tree” by clicking on the country code (e.g. SE).

11.1.3.2 Quick search

By typing a keyword in the search bar it is possible to directly find the desired case:

1. Hover over the reference number of the desired case to display a pop-up dialog.
2. Click on “Open” to open the case.

11.1.4 Upload document (from family view)

1. Click on the “Documents” tab to display the documents associated with this case
2. To upload multiple documents click on the second icon from left on the toolbar. This brings up the “Upload Documents” dialog.

3. Click on “Add Files(s)”
4. Select the files to upload.

5. Click on “Upload Files”
6. Select the title, document type and which matter/event to associate the document with. Click on “Publish Files”.

7. Files are uploaded to the Platform. Use case ends.
There are three possible starting points in the user interface. From the Outlook main menu you can log in, log out and access preferences for the plug-in.

The log in functionality is also triggered when you are trying to upload an attachment and are not logged in.
The upload functionality can also be reached from a menu in the active mail window:
Probably the most natural and smooth way to upload an attachment is to use the context menu in the mail inbox:

The context menu is only displayed if exactly one message is marked in the mailbox.
The log in window:

The user credentials are the same as when using Ipendo Platform from the web interface.
Next step is to choose an account/database from a dropdown menu:
If second layer authentication is enabled in the account, a second login window is displayed:

This dialog is the only non-modal dialog in IPOP. The reason is that you should be able to access your mail to cut & paste the password in to the password field.
If the login was successful you can now select which attachments to upload:

To select attachments the user clicks on the checkboxes corresponding to the attachments. It will as default suggest the filename (without the file extension). The document type (for Ipendo Platform) is selected in a dropdown. There is also an option to mark/unmark all attachments.
Next step is to select destination for the document. It is performed by finding a family, case or matter from a tree view:

From here you also use a search field for finding the destination object. The radio buttons can be used to filter the type of object you are interested in finding. The search result is displayed when the search button is pressed.
The search dialog:
After selecting destination either from the search result dialog or tree view the last step is to confirm the upload:
The result of the upload is displayed to the user:
Appendix D - Use cases

**UC1 Login**

Related Use Cases:

- UC2 Upload attachment
- UC3 Log out
- UC4 Change web-service host

**Description**

This use case describes the procedure of authenticating a user to the Ipendo Platform using the Outlook add-in IPOP.

**Actors**

**Primary Actor**

Ipendo Platform User

**Preconditions**

The Outlook add-in has been activated and configured in Outlook.

**Success Guarantee**

The user is authenticated against the Ipendo Platform

**Main Scenario**

The user clicks on the “IPOP” menu item.

A submenu opens and the menu items “Login to Ipendo Platform” and “Preferences” are presented.

User clicks on the item “Login to Ipendo Platform”.

A dialog is presented to the user. It contains text fields for entering “Username” and “Password” and two buttons labeled “Sign in” and “Cancel”.

The user clicks on “Sign in” after filling in username and password.

User is authenticated to the platform.

A dialog with the title “Select Client” is presented to the user. The user selects a client database from a dropdown menu and clicks on “OK”. This step will only be presented if the user has access to more than one client database.

Use case ends.

**Extensions:**

**User cancel login dialog**

At step 4 the user clicks on “Cancel”.

79
The login dialog closes.

Use case ends

**Second level authentication is enabled**
At step 6 if “second level authentication” is enabled then an e-mail is sent to the user from Ipendo Platform containing a “one-time”-password.

A new dialog is presented with a text field where the user can enter the “one-time”-password and the buttons “Log in” and “Cancel”.

User enters the “one-time” password and clicks on “Log in”.

User is authenticated to the platform.

Use case ends.

**User cancel second level authentication**
The user clicks on “Cancel” at step 3 (ext. 5.2).

The login dialog closes.

Use case ends.

**The “one-time” password expires**
At step 4 (ext. 5.2) the temporary password already expired

User is notified that “One-time password expired”. The login dialog closes.

Use case ends.

**First level authentication fails**
At step 6 if the authentication fails because of invalid username and/or password.

User is notified about entering invalid username and/or password.

User is sent back to step 4.

**Too many login attempts**
At step 6 if the authentication fails because of too many login attempts.

The user is notified that he has to contact Ipendo support.

The login dialog closes.

Use case ends.

**Second level authentication fails**
At extension 5.2 step 4 the authentication to the platform fails because of invalid password.

User is notified about entering invalid “one-time”-password

User is sent back to extension 5.2 steps 2.
**UC2 Upload attachment**

**Related Use Cases:**
UC1 Login

**Description**
This use case describes the process when a user uploads a document from the Outlook-plug-in to the Ipendo platform.

**Actors**
Primary Actor
Ipendo Platform User

**Stakeholders and Interests**

**Preconditions**
The user must be authenticated to the Ipendo Platform.

**Success Guarantee**
A document have been saved and associated with a family, case or matter in the Ipendo Platform.

**Main Scenario**
User clicks on the menu “IPOP” -> “Upload Attachment” in the active mail or right-clicks on a selected mail in the inbox to bring up a context menu.

A submenu opens and shows the menu item labeled “Upload attachment to Ipendo Platform”

The user clicks on “Upload attachment to Ipendo Platform”

A dialog is presented where all the attachment in the active mail are presented as a list. The list items are selectable by check boxes. There two buttons at the top labeled “Search” and “Browse” and two buttons at the bottom labeled “Upload” and “Cancel”.

The user selects one or more attachments. At each list item there are also a dropdown menu for “Document type” and a textbox for entering “Document title”. The user must fill out information for each checked checkbox. If the user click on “Browse” go to step 6. If he clicks on “Search” go to step 8.

A new dialog with a tree view over the root-level of the portfolio is shown. Two buttons at the bottom of the dialog: “OK” and “Cancel”. The user can navigate the portfolio tree and expanding sublevel by clicking on (+) for the parent. After selecting a family, case or matter the user clicks on “OK”. If the selected object is a case go to 7, otherwise go to 8.

A new dialog appears with the title “Select event or matter” and the user can now choose to assign the document to a specific matter or event (belonging to the selected case) via a dropdown menu. Go to 10.
A new dialog opens. It shows a search bar (textbox) on the top and radio button options to filter on “All” (no filtering) “Family”, “Case” or “Matter”. A search filter is executed when the text in the textbox changes (“type and search”). Search results are presented in a list view. At the bottom are two buttons: “OK” and “Cancel”.

The user selects an item from the search result list view and clicks on “OK”.

User is returned to the “Upload”-dialog. He clicks on the button “OK”.

A message box appears asking the user to confirm the destination and documents to upload. There are two buttons: “OK” and “Cancel”. User clicks on “OK”.

Use case ends.

Extensions:

**User cancels the upload**
At step 4 or 10 (when returned to the upload dialog) the user clicks on “Cancel”.

**The dialog closes**
Use case ends.

**Attachment is too large**
At step 5 if the attachment(s) selected by the user exceed the Ipendo Platform allowed size for document.

The user is notified and wizard’s dialogs closes.

The use case ends.