Individually Adapted Update of Software through Synchronization with a Central Configuration Database

DANIEL DANDEMAR

KTH Computer Science and Communication

Master of Science Thesis
Stockholm, Sweden 2007
Individually Adapted Update of Software through Synchronization with a Central Configuration Database

DANIEL DANDEMAR

Master’s Thesis in Computer Science (30 ECTS credits) at the School of Computer Science and Engineering
Royal Institute of Technology year 2007
Supervisor at CSC was Serafim Dahl
Examiner was Stefan Arnborg

TRITA-CSC-E 2007:119
ISRN-KTH/CSC/E--07/119--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.csc.kth.se
Individually adapted update of software through synchronization with a central configuration database

Abstract

The distribution of software has gone from diskette in the eighties to CD-ROM in the nineties and later DVD:s. The new paradigm of software distribution is to sell, distribute, update and upgrade online. In online software distribution it is important to be able to keep track of the licenses and their relation to other licenses. A configuration is built up by licenses that relate to each other.

This Master’s thesis will look at how you can design a software distribution system. A software distribution system consists of three different parts; a database, a server that acts as an interface to the database and a client that communicates with the server. There’s two different approaches; either the software is always complete on the user’s machine or only parts that is needed on the user’s machine is downloaded and kept up to date. In this Master’s thesis the second approach is used, though this approach will also cover the first case.

The core of the problem is to find a good database structure; the Customer Configuration Update (CCU) method from Utrecht University contains a good meta-data model that can be part of the foundation. The project at Utrecht is an ongoing research that wasn’t finished when this Master’s thesis was written.

Software distribution system turns out to be quite powerful since it will let a user install a base version on a new computer then enter a license key and all the missing parts will be downloaded and installed on the new machine. Also if an add-on is bought then the files needed for that add-on will be downloaded and installed, even on a new machine since the user’s configuration is stored on the server.
**Individuellt anpassad uppdatering av programvara genom synkronisering mot en central konfigurationsdatabas**

**Sammanfattning**


Mjukvarudistributionssystem visar sig vara väldigt kraftfulla. En användare kan installera produkten på en dator och därefter ange licensnumret, vilket kommer att innebära att de delar som saknas automatiskt hämtas och installeras. Likaså om användaren köper tilläggstjänster som kräver nya filer så hämtas dessa automatiskt. Detta gäller även om användaren senare installerar om produkten på en ny dator.
# Contents

1  Introduction.......................................................................................................................... 1  
1.1  Background .................................................................................................................... 1  
1.2  Problem .......................................................................................................................... 2  
1.3  Purpose ............................................................................................................................ 2  
1.4  Delimitations .................................................................................................................. 2  
1.5  Summary .......................................................................................................................... 2  
2  Theory .................................................................................................................................. 3  
  2.1  Distribution model ........................................................................................................... 3  
    2.1.1  Customer Configuration Updating ........................................................................... 3  
  2.2  Design pattern ................................................................................................................ 4  
    2.2.1  Patterns for software licensing ............................................................................... 4  
  2.3  Server ............................................................................................................................. 6  
    2.3.1  Web service ............................................................................................................. 6  
  2.4  Signature and secure communication .......................................................................... 7  
    2.4.1  Hash functions ....................................................................................................... 7  
  2.5  Hardware signature in practice ...................................................................................... 9  
    2.5.1  Microsoft Windows XP Activation - signature ...................................................... 9  
  2.6  Summary .......................................................................................................................... 10  
3  Overview of problem .......................................................................................................... 11  
  3.1  Model ................................................................................................................................ 11  
  3.2  Summary .......................................................................................................................... 12  
4  Design decisions ................................................................................................................... 13  
  4.1  Structure .......................................................................................................................... 13  
  4.2  Partial and full product .................................................................................................... 13  
  4.3  Model ................................................................................................................................ 13  
  4.4  Database .......................................................................................................................... 14  
  4.5  Communication .............................................................................................................. 14  
  4.6  Server ............................................................................................................................. 14  
  4.7  Validation of downloads ............................................................................................... 14  
  4.8  Validation of license ...................................................................................................... 14  
    4.8.1  Client ....................................................................................................................... 14  
    4.8.2  Server ..................................................................................................................... 14  
  4.9  Summary .......................................................................................................................... 15  
5  Method .................................................................................................................................. 16  
  5.1  OMT and OOSE .............................................................................................................. 16  
    5.1.1  Requirements elicitation ......................................................................................... 16  
    5.1.2  Requirement analysis ............................................................................................. 16  
    5.1.3  System Design, Object Design ............................................................................. 17  
  5.2  Summary .......................................................................................................................... 17  
6  Requirements Elicitation ...................................................................................................... 18  
  6.1  Functional requirements ............................................................................................... 18  
  6.2  Non functional requirements ......................................................................................... 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.3</td>
<td>Future work</td>
<td>48</td>
</tr>
<tr>
<td>11.3.1</td>
<td>Multiuser license</td>
<td>48</td>
</tr>
<tr>
<td>11.3.2</td>
<td>Online business</td>
<td>48</td>
</tr>
<tr>
<td>11.3.3</td>
<td>Validate upgrade</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Appendix A</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Appendix B</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>Appendix C</td>
<td>57</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background

The Software Industry has in its brief history several times changed its way of distributing programs. In the eighties when the first personal computers hit the market software vendors mostly distributed programs on diskettes. In the nineties the CD-ROM became the most used media for distribution. Now at 2007, 18 years after the invention of the world wide web by Sir Tim Berners-Lee and Robert Cailliau, a huge number of computers are connected to the Internet and large amounts of data is distributed online for updates, program purchases and software downloads.

Effective Studios is a software company whose ambition is to create easy-to-use programs targeting Small and Medium Enterprises (SME). Its mission is to "Simplify Internet Participation". Among the products is the award-winning software Site Studio, which has users all over the world. Internet is being used as the major distribution channel. Only in retail - a boxed CD-ROM is available.

The new paradigm of software distribution is to sell, distribute, update and upgrade online, and to know exactly to whom every instance of a program belongs. It is also important to know which upgrades are possible for an instance and to create customized promotional offers to customers.

Business goals for online distribution systems:

- Online purchasing (anonymous users are turned into customers)
- Zero administration (customer automatically receives purchased software)
- Online validation (prohibit illegal duplication)
- Online content (prohibit illegal duplication)
- Purchase history (the product can be rebuilt online if required)
- Customized offers (per-instance offers generated from user behaviors)

Purchases can be divided into three categories:

- Buy-out (customer buys and owns the license forever)
- One-Shot (the customer buys a voucher which can be traded and used once)
- Subscription (customer buys a time-limited license which is renewed periodically)

Technical goals:

- Track downloaded software (trials)
- Customize offers for trials
- Update trial into full version software
- Keep instance data, so upgrades can be determined
- Distribute upgrades
- Disable expired upgrades
1.2 Problem

This Master’s thesis is all about best practices for building an online software distribution system that fits the needs of Effective Studios and outline Effective Studios online distribution system.

Parts that need to be solved are

- Store user configuration
- Distribute user configuration to user’s machine
- Update user configuration on server (i.e. Add/remove add-ons)
- Restrict access to online resources
- Secure communication regarding license number
- Find a suitable distribution of files. Packages, single files or mix
- Collect information about the usage of the product
- Activate product
- Activate add-on
- Online validation

1.3 Purpose

The purpose is to fulfill the following goals.

Business goals for online distribution systems:

- Zero administration (customer automatically receives purchased software)
- Online validation (prohibit illegal duplication)

Technical goals:

- Customize offers for trials
- Update trial into full version software
- Keep instance data, so upgrades can be determined
- Distribute upgrades
- Disable expired upgrades

1.4 Delimitations

These areas are not covered in this Master’s thesis.

- The purchasing process.
- Validation of downloaded files and download mechanism.

1.5 Summary

We have now described the problem and the purpose that this Master’s thesis will try to solve. In the next chapter we will do a research in these areas. We will then make a brief overview of the problem and solve some parts based on the researched that has been done. After that we will describe the method used to solve the problem of the design and implementation in a structured way and also apply that method. Finally, we will summarize what we have done and draw conclusion of our work.
2 Theory

This chapter will describe the interesting parts of the information that was gathered during the research of information of the problem area.

First we will describe a method for distributing update that is being developed at the Utrecht University. The model describes the steps release, delivery, deployment, activation and usage. After that we will take a look at two design patterns that can be used to lock the software. The first shows how to lock the software after a certain time period; the second one shows how to lock the software to a specific computer. We then explore Web Services that can be used in client server solution and will see if they have any advantages.

Hash functions are an important area and are used in digital signatures and encrypted communications; we will take a brief look in this area. We finish the chapter by exploring how Microsoft applies hash functions to lock a license to a specific computer.

2.1 Distribution model

2.1.1 Customer Configuration Updating

Customer Configuration Updating (CCU) is a production method that is currently being developed by researchers at Utrecht University (1).

The customer configuration updating is defined as the combination of the vendor side release process, the product or update delivery process, the customer side deployment process, and the activation and usage process (2).

The four steps (release, delivery, deployment, activation and usage) in the CCU model are shown in Figure 2-1.

![Figure 2-1 CCU model](image)

A process model for the delivery and activation and usage has been published and we will take a closer look at those two.
2.1.1.1 Delivery phase

The delivery process data model is displayed in Figure C-1 (see Appendix C). On the left side is the meta-process model with all the activities for this phase and on the right side is the meta-data model.

When a package is complete it is deployed to a repository and the customers are then informed about this. The package consists of the following five elements (as shown in the meta-data model Figure C-1): software package, system description, manual, license, management information. The software package contains software components and the version and forms a product. System description describes what components are included and the requirements requested; it also contains information on how the product is related to other products already installed. Manual is a document describing how to use the software. License is the license agreement for the software. Management information is a document that describes how to administrate the system.

The customer relationship management system (CRM) contains information about the customers.

Software configuration management contains information about the products and the configuration that a customer has. It also contains information about the current version, what updates the user has installed, the operating system used, special modifications for the client that is needed and feedback collected from the user. The feedback is data such as bug reports, product usage data, error reports and usage questions.

2.1.1.2 Activation and usage phase

The activity process data model is displayed in Figure C-2 (see Appendix C) (3). As with the delivery process model the left side is the meta-process model and on the right side is the meta-data model. At the start of the activation the activation request can either be a new activation request or a renewal request. The customer is then identified and the contract for the customer is located (the contract can contain several software licenses). If no license exists for the product or part of the product a new license can be generated (EULA). If it is a new or a renewal of a license a license code is generated. The license code consists of letters and numbers and is used to identify the customer and the contract. The license is then encoded and the license is stored. When the activation or re-activation occurs a unique installation id is generated from the hardware signatures. And finally the license and the code are sent to the customer.

2.2 Design pattern

2.2.1 Patterns for software licensing

Kaminski and Perry (4) have presented patterns for software licensing. In their paper they presented 18 patterns regarding this issue and these are the two patterns that are the most interesting for this Master’s thesis. The patterns presented in the paper (4) can sometimes be mixed with each other to fulfill the need. The two patterns presented below (Time- Based and Node-locked) can be mixed.

2.2.1.1 Time- Based Software License

Context: “Sometimes there is a need to have an application that should be used only for a specific period of time. It is important to prevent the customer from running the software after the agreed period of time.” (4) See Figure 2-2 and Figure 2-3.

The time could either be a fixed date or an amount of time that is consumed.

Time-Based can be useful for demo products (trials) and for companies that request a product for a short period of time and therefore don’t want to buy it.
2.2.1.2 Node-lock (named host) Software License

Context: “It is desirable to have software that is machine dependant. It may not matter who is using the application or how many user are there, but it is crucial that the application is installed and runs on a specific computer.” (4) See Figure 2-4 and Figure 2-5.

The hardware signature can be created from a either a single hardware id or a combination of ids i.e. CPU, HD, NIC, motherboard, graphic card and other devices. See the hardware signature section.
2.3 Server

In the Master’s thesis we need to have a server that the client can communicate with. The server will contain the logic and interact with a persistent media. In this Master’s thesis we will implement the server using web services.

2.3.1 Web service

One advantage with Web services is that it is supported by major companies like IBM, Microsoft and BEA. Also it uses open technology (XML) and standards that is managed by WRC and OASIS. Web services support interoperable communication between different Operating System. They are also loosely coupled and are self describing (using WSDL).

2.3.1.1 SOAP

Web services use the SOAP protocol for exchanging the XML messages (5). The messages can be sent via various transport protocols; such as HTTP/HTTPS, SMTP etc.
An advantage is that the messages can be sent via HTTP and thereby pass through most firewalls.

One of the negative aspects is that the communication using XML takes up more space than binary communication (such as CORBA and RMI).

2.3.1.2 WSDL
WSDL (Web Services Description Language) is used to describe the Web service and uses XML (5).

WSDL describes what a service does; the methods the service provides and the return value. It also describes where the service is located and how the service should be accessed.

2.3.1.3 UDDI
UDDI (Universal Description, Discovery, and Integration) is a service directory that provides information about business and services (5).

Information about a web service can be published in this directory and other i.e. services can then use this directory to look up and retrieve information about the service. Where it is, what types of messages is used etc.

UDDI has three parts:
White pages that contains contact information about the company.
Yellow pages that contains classification information about the service.
Green pages that contains information about how to invoke the service.

2.4 Signature and secure communication

2.4.1 Hash functions
To hash something is to divide it into small parts. A hash function is said to be one-way if it is computational infeasible to determine the message m if you are given the hash of h(m) (6).

If two messages have the same hash value then the pair is called a collision. A collision resistant hash function is a hash function that given the hash value of function h(m) it is computational infeasible to determine any other message m* with the same hash value, h(m) = h(m*).

Cryptographic hash functions are used to “fingerprint” a message (7) often referred to as message digest.

Hash functions are also used in digital signatures.

2.4.1.1 SHA - Secure Hashing Algorithm
SHA-1 processes 512-bit blocks and generates 160-bit hash values(6) which form the message digest.

Chinese researchers have found security flaw in SHA-1(8)(9)(10) which break the non-repudiation part which cryptographers rely on. The research describes how to create collisions with SHA-1 algorithm 2000 times faster than before (with brute force). The number or tries can still take 1000 years for a single personal computer.

NIST (National Institute of Standards and Technology) has planned to end the federal usage of SHA-1 by the end of 2010.

SHA-2 is a family that consists of SHA-224, SHA-256, SHA-384 and SHA-512 where the number indicates the number of bits in the digest. It is likely that SHA-2 will replace SHA-1. At the moment SHA-2 isn’t supported on all OS yet and CAs is not using it.
2.4.1.2 MD5 – Message-Digest algorithm 5
Generates a 128 bit message digest.

The MD5 also has security flaw. Vlastimil Klíma has a report(11) and source code(12) on MD5 collision at an average of 17 seconds on a Pentium 4 (3.2 GHz).

Performance(6) shown in Table 2-1.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Million cycles/byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>7-8</td>
</tr>
<tr>
<td>SHA-1</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 2-1 Performance Measurement for Hash Functions

2.4.1.3 MAC
MAC (Message Authentication Code) is a keyed hash function that takes a secret key and a message and produces a message digest (Wikipedia). MAC values are generated and verified with the same secret key.

2.4.1.4 Digital signatures
In contrast with MAC digital signatures use asymmetric encryption. Asymmetric encryption algorithms use a different key for encryption and decryption. The keys are related to each other but you can’t derive one key from the other key by analysis. One of the keys is usually called public key while the other is the private key. The public key is used for decryption and it can be distributed to other users. The private key is used for encryption and should be kept secret and stored in a safe place (i.e. a smart card).

Usually when creating a digital signature a hash function is applied to the file which creates a message digest. The message digest is then encrypted using the private key which creates the digital signature. To validate the user uses the same hash function on the file and then decrypt the digital signature using the public key. Then by comparing the values the user can tell if the file has been tampered with.

Digital signatures support non-repudiation; the sender can’t claim that she didn’t send the information. It also supports integrity; the message can be validated that it hasn’t changed.

2.4.1.5 Digital certificate
Certificates are part of PKI (Public-Key Infrastructure) which is a secure system that manages and controls digital certificates (7).

The digital certificate consists of three things (13):

- A public key
- Certificate information
- One or more digital signatures

Web browsers usual contains root CAs, trusted CAs.

Digital certificates are sold by companies such as Verisign, Thawte and GlobalSign.

2.4.1.6 SSL
SSL (Secure Socket Layer) uses PKI.
Two users (client/server) connect and agree on what cryptographic algorithm they should use (7). The server sends over its certificate to the client which is signed by a CA (Certification Authority). The certificate is then validated on the client computer using the CA’s public verification key. The client then generates a master key which is encrypted with the server’s public key (that is received from the server’s certificate) and sent to the server. The client and the server then generate two keys from the master key. One of the keys is used to authenticate the messages sent and the other is used to encrypt the communication.

SSL certificates can be bought by i.e. Verisign, Thawte and GlobalSign.

### 2.5 Hardware signature in practice

#### 2.5.1 Microsoft Windows XP Activation - signature

In Microsoft Windows XP the product activation uses hardware ids to generate a signature (14)(15). This way the installation is bound to that computer. The hash that is created is made up of 10 different hardware ids from the user’s computer. Each of the 10 hardware ids are hashed in separate, then only a few bits from each hash is used in the final signature to ensure the users privacy. Due to the short length of bits used in each hash many components on different computers will end up with the same hash. As an example in the IDE case we use 4 bits which will result in only 16 different possible values. The final signature with all the hashes and the “dockable” flag as well as the hash version combined will make up 64 bits.

Table 2-2 shows the hardware components used

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Number of bits in the hashed value used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Adapter</td>
<td>5</td>
</tr>
<tr>
<td>SCSI Adapter</td>
<td>5</td>
</tr>
<tr>
<td>IDE Adapter</td>
<td>4</td>
</tr>
<tr>
<td>Network Adapter MAC Address</td>
<td>10</td>
</tr>
<tr>
<td>RAM Amount Range</td>
<td>3</td>
</tr>
<tr>
<td>Processor Type</td>
<td>3</td>
</tr>
<tr>
<td>Processor Serial Number</td>
<td>6</td>
</tr>
<tr>
<td>Hard Drive Device</td>
<td>7</td>
</tr>
<tr>
<td>Hard Drive Volume Serial Number</td>
<td>10</td>
</tr>
<tr>
<td>CD-ROM</td>
<td>6</td>
</tr>
<tr>
<td>“Dockable”</td>
<td>1</td>
</tr>
<tr>
<td>Hardware Hash version</td>
<td>3</td>
</tr>
</tbody>
</table>

*version of algorithm used*
Table 2-3 shows the how the RAM is counted

<table>
<thead>
<tr>
<th>Bit value</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(unused)</td>
</tr>
<tr>
<td>1</td>
<td>&lt; 32 MB</td>
</tr>
<tr>
<td>2</td>
<td>32 &lt;= x &lt;= 63 MB</td>
</tr>
<tr>
<td>3</td>
<td>64 &lt;= x &lt;= 127 MB</td>
</tr>
<tr>
<td>4</td>
<td>128 &lt;= x &lt;= 255 MB</td>
</tr>
<tr>
<td>5</td>
<td>256 &lt;= x &lt;= 511 MB</td>
</tr>
<tr>
<td>6</td>
<td>512 &lt;= x &lt;= 1023 MB</td>
</tr>
<tr>
<td>7</td>
<td>1024 &lt;= x</td>
</tr>
</tbody>
</table>

Table 2-3 Bits set for RAM values

A check is made with the current signature when the user logs in to the system to see that the hardware hasn’t changed too much.

If a network card is present then the user can change 5 of the items and the system will be counted as valid.

If a network card is not present (or changed) then the user can change up to 3 items (including the network card) and the system will still be valid.

If the system is “dockable” there is a slight change in these values but the method is the same.

The first registration is kept so for instance changing the display adapter A to B then to C will count as one change.

The hash value together with a product id is then sent to the server during the activation process.

### 2.6 Summary

In this chapter we have looked at a method from the Utrecht University that is currently being developed. We then looked at design patterns for locking a license on a specific machine and limiting software based on time. We took a look at Web Services which supports interoperability between operation systems and relies on standards that is managed by big organizations like W3C and OASIS. For this Master’s thesis we are interested in using Web Services since they are firewall friendly. We looked at hash functions and digital certificate and at the end of the chapter we explored how Microsoft uses hash function on 10 hardware ids to create a signature of the user’s computer.

In the next chapter we will get an overview of the problem.
3 Overview of problem

Before we start we need to get an overview of what needs to be done. This chapter depicts the problem as a flow of steps that can occur when running a product.

3.1 Model

The usual product flow for trial and full users is shown in Figure 3-1.

Figure 3-1 Product flow

Purpose of each step:

- Step 1, Start product
  User launches the product.
- Step 2, Validate license
  Verify that the license is valid either. The validation can be on the user’s machine or on the server depending on the license.
- Step 3, Automatic update of user’s configuration on user’s machine
  The user configuration is updated on the user’s machine to reflect the new state if any changes have been done since the last update. The configuration that the user should have is stored on the server.
- Step 4, Get offer
  Present an offer to a specific customer or a group of customers. The offer is retrieved from the server.
- Step 5a, Evaluate product
  Run the product with a trial license.
- Step 5b, Buy product
  User buys the product and receives a license number. The purchase is done on a website.
- Step 6, Activate product

Step 7, Use online resource

Step 8, Run product

Step 9, Buy add-on

Step 10, Activate add-on

Step 11, Evaluate product

Step 12, Get offer

Step 13, Buy product

Step 14, Activate product
The user uses the license number to activate the product from being a trial to a full version.

- Step 7, Run product
  Run the product with a full license.

- Step 8, Use on-line resource
  Access on-line resources. The product needs to access content on a server and a validation of the license needs to occur.

- Step 9, Buy add-on
  User buys an add-on to the product and receives a license number for the add-on. The purchase is done on a website.

- Step 10, Activate add-on
  The user uses the license number to activate the add-on.

In this Master’s thesis we will look at step 2, 3, 4, 6, 8 and 10 as depicted in Figure 3-2 below. Step 1, 5a, 5b, 7 and 9 is outside the scope of this Master’s thesis. In the future step 5 and 9 could be included in the Software distribution system, but for now these two steps is performed on a website.

Figure 3-2 Product flow, areas covered in this Master’s thesis is marked with dotted lines

### 3.2 Summary

We have now got a brief overview of what parts in the process that we will look at. In the next chapter we will make design decisions based on what we know.
4 Design decisions

In this chapter we will make some early design decisions based on what we know from the requirements, the product flow and the research that we have looked at. We will revise the design decision at the end and modify some of them.

4.1 Structure

At this point we also have an overview of the structure as shown in Figure 4-1. The mechanism on the client needs to be a module that can be used in the products in the feature. We will refer to the module as that we will refer to as Software licensing system. We also have a server that will perform logic and interact with a persistent media.

![Figure 4-1 Basic structure](image)

4.2 Partial and full product

There are two different approaches.

Either the product is always complete, with all the modules that are needed for a full version (and the add-ons) even if the user only runs a limited version. The functionality is then activated through licenses, much like flags.

Or the product only contains the modules that are needed for the licenses that are active. The first approach is easier to manage since an update always updates the complete system, the downside is that it is harder to integrate new modules and that the size of the product is unnecessary big.

The other approach makes it harder to keep the system synchronized since a structure is needed to keep track of the installed modules and their version. But it is easier to add new modules to the product and the size of the base product is smaller.

In this Master’s thesis we will use the second approach. This way if a module is updated and the exe hasn’t changed then only the users that need that module will retrieve an update. The distribution model developed in this way will also support the first approach since the base product can contain a complete package and the modules (add-ons) will only have a license key and no files attached to it.

4.3 Model

We need to find a process that from a request validates the correctness of that and returns the value. After studying the CCU method I have concluded that it will be a good foundation to start from. The method shows the flow from a release of a distribution to the deployment at the user’s machine. The two models that I looked at (delivery and activation) include meta-data models which will be a good foundation to have as a backbone. We will make modifications so that it will fit our needs. The modification that we will apply is described when we solve the problem. At this point we only conclude that we will be using parts of it.
4.4 Database

The database will have three parts

- Repository - structure that store the updates and full versions.
- License Data - structure that store the configuration that a customer has, that is what licenses the customer has and what is their state. It will also include product info and customer info.
- License Usage - structure to store the usage information. That is the information about the usage of the software. It can be when the product was started and terminated, when was it installed, what resources has been accessed and so on.

4.5 Communication

We need a secure communication. This can be achieved through SSL. So this is a small issue and all we will need to do is to install a SSL certificate on the web server at the end to achieve this. So this issue can be omitted during the development phase.

4.6 Server

We want to minimize the risk of being blocked by firewalls and we also want to communicate using a standard. This is since the target is companies and therefore many of them is likely to have a firewall. If we use Web services on the server side using SOAP messages sent over HTTP then we will reduce that risk.

4.7 Validation of downloads

Validation of downloaded files to ensure that they are from the company is needed. By applying a digital signature on the exe file we will make the process of verifying this easy. So the design decision on the distribution type (files, packages etc) will be that a single signed exe file will be distributed on every update (or full product download). As mentioned in the delimitations the actual validation of the file is done by the product and is not included in the Master’s thesis.

4.8 Validation of license

We need to protect the usage of the software somehow. We’ll keep the patterns described in the theory chapter in back of mind when we have to address this issue. The basic idea is that we have two different points where we perform the check, on the client and on the server.

4.8.1 Client

The client only validates against time which is normal in a trial version. If a full version can’t connect to the server then the same mechanism will be used as the one for the trial version. The time limit will then be set to 60 days, if a connection can be made within that amount of days then the days is set to 60 again. If no connection has been made within 60 days then the user will be prompted that a connection is needed and the product will quit.

4.8.2 Server

On the server side we’ll have both time based in the case of a time limited license (such as the add-on Templates that is being used in Site Studio 6) and node lock. The node lock will allow up to x different nodes, this will ensure us that the license won’t be distributed to a broad crowed. By using the patterns we can add new features in the future in an easy way such as add
a time consumer instead. To achieve the node lock we will use a hash function on the network card address and since MD5 is twice as fast as the SHA-1 algorithm we will use MD5 for this. Although there is possible to create collisions in the MD5 (and SHA-1) there is no need for us to use another algorithm than MD5 since we will only use it to create the customers hardware signature. The approach Microsoft has in XP is to use 10 ids (see the Theory chapter) from the user’s computer and only use a portion of each in the final result. This was to ensure privacy and the different ids had different weights. In our solution we will only use one id to generate the signature, the NIC. The positive is that the user can upgrade all the components on the computer, except the NIC. The downside is that if the user changes the NIC and that is the only thing that has changed then it will be recognized as a new instance. If this would tend to be a problem in the future then this can easily be changed.

4.9 Summary

We have made some early design decisions based on the requirements and the research that we have looked at. We will now start the development of the software. In the next chapter we will describe the method used.
5 Method

In this chapter we’ll describe a method that is a mix of Object Modeling Technique (OMT) and Object Oriented Software Engineering (OOSE) that we used to solve the problem. It has three parts; requirements elicitation and analysis, system and object design and implementation. We applied this method to solve our problem.

5.1 OMT and OOSE

Since this is a software engineering problem the method used is OMT and OOSE is used. The methodology used is a mix of OMT and OOSE described in the book *Object-Oriented Software Engineering* (16). Three steps were performed:

1. Requirements elicitation and analysis
2. System and object design
3. Implementation

In order to get a picture of what the problem really consisted of step one was performed. This step was also repeated along the way when more facts about the problem were needed.

5.1.1 Requirements elicitation

Requirements elicitation includes the following activities:

- Identifying actors
- Identifying scenarios
- Identifying use cases
- Refining use cases
- Identifying relationships among use cases
- Identifying nonfunctional requirements

The purpose of the use case and scenarios is to define requirements that are validated by the user early in the development process.

5.1.2 Requirement analysis

Requirement analysis produces a model of the system (analysis model). The analysis model consists of three models:

- Functional model
- Analysis object model
- Dynamic model

The functional model is represented by use cases and scenarios. The analysis object model is represented by uses class and object diagrams. The dynamic model is represented by statechart and sequence diagrams. An overview of the structure is shown in Figure 5-1 and Figure 5-2.
Entity, boundary and control objects are identified. Where entity objects is the persistent information, boundary objects are the interaction between the actor and the system and the control objects realizes the use case.

Sequence diagrams ties the use cases to objects.

### 5.1.3 System Design, Object Design

The mapping and translation to code is only touched briefly in the report since it is not that interesting. Instead the prototype is displayed and discussed. But in the process we would have identified sub systems using heuristic and minimized the number of associations crossing the boundaries in the subsystems. The parts on the client side in our case will be in a new sub system separate from the original sub systems in the client. The module will be called LicenseModule and can be reused in other products.

### 5.2 Summary

We have described the method that is used to solve the problem in a structured way. In the next chapters we will apply the method that we described. The first step is the requirements elicitation and analysis.
6 Requirements Elicitation

In this chapter we will apply the first step of the method previously described, requirements elicitation.

6.1 Functional requirements

The functional requirements are listed in chapter 1 (purpose).

6.2 Non functional requirements

- The communication with a persistent media must use transaction if changes are to be made.
- The client part should run on Microsoft Windows XP SP 2 and above.
- The server part should run on a Microsoft Environment such as Windows Server 2003.
- If IIS is used then that part should run on version 6 and above.
- Database should be stored on Microsoft SQL server.
- Microsoft Visual Studio 2005 should be used as the developing environment. C++ and/or C# can be used.
- Communication with servers should be secure.
- The user must be able to verify that the downloaded installation file hasn’t been changed by another party.
- It should be a module that can be included in our products.

6.3 Actors

In the chapter Design decisions we got an overview of the structure (Figure 4-1). The major part of the Master’s thesis is the server. The module on the client side will be called Software licensing system and the server part will be called Server. The user and product are seen as one actor.

We have the following actor.

- Product

As mentioned in the design decisions we have the Software licensing system and a server. Thus this gives us the following connection (Figure 6-1).

Figure 6-1 The actor product communication with the Software license system which in turn communicates with the server.
6.4 Scenarios

- Update configuration
- Evaluate product
- Run product
- Use on-line resources
- Activate add-on
- Activate product

See Appendix A for scenarios.

6.5 Use cases

We have identified the following five use cases together with a brief explanation about each use case.

- Validate license (Table 6-1) – First a connection to the server is made and if successful then the validation is performed on the server, this is done through the get licenses. The trial version also tries this approach, but the server can choose to ignore the request and then a local check will be performed. If no connection to the server can be made then the license is checked local. The software will run for 60 days (if a full version) on the current settings if no connection can be made, after that time the software will display a message to the user and stop working until a connection can be made. On the local license check the Time-based pattern is used. When connected to the server the Time-based and Node-lock pattern is used. This corresponds to step 2 in the model (Figure 3-2).

- Automatic update of user’s configuration on user’s machine (Table 6-2) – Get the licenses that need to be updated and their location. This corresponds to step 3 in the model (Figure 3-2).

- Get offer (Table A-4) – Get a specialized offer for the user. This corresponds to step 4 in the model (Figure 3-2).

- Activate add-on (Table A-5) – Activate and bind an add-on license to the base license (product license). This corresponds to step 10 in the model (Figure 3-2).

- Use on-line resource (Table A-6). This corresponds to step 8 in the model (Figure 3-2).

<table>
<thead>
<tr>
<th>Use case name</th>
<th>Validate license</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participating actor</td>
<td>product</td>
</tr>
<tr>
<td>Flow of events</td>
<td>1. The product request that the license is validated. A request is sent to the Software Licensing System to perform a validation.</td>
</tr>
<tr>
<td></td>
<td>2. The Software Licensing System retrieves the license from the registry.</td>
</tr>
<tr>
<td></td>
<td>3. If it is a trial license the Software Licensing System retrieves the date of installation that is stored in the registry. It then checks if the number of days has been exceeded (30 days). If it has been exceeded then a notification about that is returned to the product.</td>
</tr>
<tr>
<td></td>
<td>4. If it is not a trial license the Software Licensing System tries to send a request to the Server with the license number for the product and the hardware signature. If no connection then step 3 is performed but the days are increased to 60 days and the last connection date is retrieved from the registry. Every time a</td>
</tr>
</tbody>
</table>
Successful connection is made the connection date is updated in registry.

5. The Server receives the request and retrieves the status of the license from the persistent media. The status can be (inactive, active, expired and locked). If the license is inactive then the status is updated to active in the persistent media. If the license is active then an ok status is returned, in case of a subscription license then expire date is also returned. The licenses attached to the license sent in are also returned as well as their status. Otherwise a failed status is returned with the reason.

6. The Software Licensing System processes the result and if the response is ok then the registry entry that keeps the last connection date is updated. The response is then returned to the client.

7. If there is an ok response then the product continues to start. Otherwise the user is informed about the failure and the product ends the execution.

<table>
<thead>
<tr>
<th>Use case name</th>
<th>Automatic update of user’s configuration on user’s machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participating actor</td>
<td>Product</td>
</tr>
<tr>
<td>Flow of events</td>
<td>1. The product checks for update to see if there are any new updates on the server or if the user’s configuration has changed. A request is sent to the Software licensing system requesting the user’s machine to be updated.</td>
</tr>
<tr>
<td></td>
<td>2. The Software licensing system sends the base license (product license) and all the other licenses that is active on the client to the server. Information about the current licenses and their version is also sent.</td>
</tr>
<tr>
<td></td>
<td>3. The Server receives the request and validates the base license and retrieves all the licenses that are bound to the base license.</td>
</tr>
<tr>
<td></td>
<td>4. The server then retrieves a list of the latest possible version (it doesn’t need to be the latest version since it can be several updates before the last one is achieved) that the license should have or update to.</td>
</tr>
<tr>
<td></td>
<td>5. That list is then combined with the matches from the user’s request (thus filtering out licenses that shouldn’t be in the list). The new list is then returned to the Software licensing system. The returned information contains the status of the license (active, expire date if subscription, locked). The new version number and where it can be downloaded (URLs) is also sent back. If there is no new update then the same version is returned back but no URLs are added.</td>
</tr>
<tr>
<td></td>
<td>6. The Software licensing system processes the result and if there’s a later version of any of the components or a new one then it is added to a list together with the URL(s) where it is located. The list</td>
</tr>
</tbody>
</table>
is then returned to the product.

7. The returned list is compared to the current licenses. If there is a different then the user’s configuration is updated. If a new license exists then it is added, if a license isn’t in the list then it is removed. The license expire date of licenses is also updated if there exist one (subscription licenses).

8. If there is any updates that needs to be run the product informs the user that there’s updates that the configuration needs to be updated. If the user wishes to update then the files are downloaded.

9. When all files are downloaded the files is installed. After each installation the current version of the license current version is updated in the registry. If the installation needs to be restarted then a restart is performed and the installation process continues with the next file.

10. When all updates in the list have been installed step 2 is performed again since there could be more updates (incremental updates).

11. If no more updates are available then the update of the user’s configuration is complete.

Table 6-2 Use case. Automatic update of user’s configuration on user’s machine.

For more use cases see Appendix A.

6.6 Summary

In this chapter we performed the requirement elicitation and made use cases for all the functionality that should be supported. It gave us a good view of what should be done and how the flow is. In the next chapter we will perform requirement analysis that will help us identify the objects, and their behavior, relationships, classification and organization.
7 Requirement analysis

In this chapter we will perform requirement analysis. At the end of the chapter we have identified the objects and their behavior, relationships, classification and organization. The analysis model is used together with the nonfunctional requirements to prepare for the architecture of the system during high-level design (16).

For entity, boundary and control objects see Appendix B.

7.1 Sequence diagrams

7.1.1 Validate license

Figure 7-1 shows the sequence that occurs when the user validates the license. The product calls retrieves the license to be validated from the registry then the ValidateLicense() in the softwareLicenseSystem module is called. A validateLicenseControl is created. The validateLicenseControl is the manager that contains the logic for the validate process. If it is a trial then a local time check is performed. This is performed with the time-based pattern described in the theory chapter. The result is returned to the product and if the time limit has expired then the user receives information about this and the product terminates. If it is a full version then a validation request is sent to the server. The server validates the license and returns the result. The result contains a return code indicating if the call succeeded or not as well as a message explaining the failure in case of a failure. Also the license status is returned (active, inactive, expired, locked or invalid).

Figure 7-1 Sequence diagram, validate license.
7.1.2 Automatic update of user’s configuration on user’s machine

Figure 7-2 shows the process of automatic update of user’s configuration on user’s machine. The product calls the Software licensing system that created the updateControl that contains the logic. A call is then sent to the Server, the call contains the current licenses on the user’s machine and their version. The Server validates the base license to make sure that the request is valid. The next possible update is retrieved for the licenses that the user should have. This next possible version is based on the version that was passed with the request. The result is then passed back to the product that checks if there are any updates that are needed. In that case they are downloaded and installed. The product might need to restart, in that case the update process continues after the restart. When the update is complete the registry is updated and the product is restarted.

![Sequence diagram of the automatic update of user’s configuration on user’s machine]
7.1.3 Get offer

Figure 7-3 show the sequence diagram from Get offer. The product retrieves the license number from the registry and then calls the Software licensing system which then creates the offerControl. The offerControl has the logic for the get offer process. The offerControl gets the hardware signature of the user’s machine. The license and the signature are passed to the server that retrieves the offer for the license if any. If an offer is found then it is returned to the offerControl. The return value is then passed back to the product that displays the offer if one was found otherwise a default offer is displayed to the user.

![Sequence diagram, Get offer](image)

7.1.4 Activate add-on

Figure 7-4 shows the sequence diagram for Activate add-on. The software licensing system is called to activate an add-on. An activateAddonControl is created that contains the logic for the activation steps. A request to activate the add-on is sent to the Server. The request contains the base license (product), the hardware signature and the add-on to activate. The Server then tries to activate the add-on. The result is then passed back to the product.
7.1.5 **Use on-line resource**

The product needs to access an on-line resource (Figure 7-5). The hardware signature is retrieved from the Software licensing system. The base license together with the hardware signature is then passed to the Server. If the request is successful then the resource is returned.
For more sequence diagram see Appendix B.

**7.2 Summary**

In this chapter we managed the requirements elicitation and analysis which resulted in the specification of the system that the customer understands and a model that the developer uses. In the next chapter we will perform the next step, system design. This will let us transform the previous model (analysis) into a system design model.
8 System design and object design

In this chapter we will perform system design and object design.

8.1 Client

The license parts on the client are added to a module that is integrated in the product. The license module uses the façade patterns to shield off the internal knowledge from the outside users as shown in Figure 8-1.

![Figure 8-1 Software licensing system inside the product.](image)

8.2 Server

The parts in the Server have no associations with each other (except for the storage) and are therefore decomposed into four parts as shown in Figure 8-2. These parts are then implemented as Web Services and the separate parts will make it easier to manage the scalability in the future.

On the server side we have four Web Services that the product will communicate with. To reduce the risk of being blocked by firewalls the registration and activation is implemented using web services. The xml is sent over http on port 80 using SOAP.

Note that the return values is actually wrapped inside an object then returned.

![Figure 8-2 Web Services](image)
8.2.1 LicenseService
LicenseService manages the communication that has to do with the license.
The LicenseService has the following functions.

8.2.1.1 ValidateLicense
Validates that the base license of the product is valid and returns the status of the license.

In  | HID | Hardware id
| BaseLID | Product license number
Ret | ReturnCode | Ok, error
| Message | Return message
| LicenseStatus | Active, Inactive, expired, locked, invalid

8.2.1.2 ActivateAddon
Activates an add-on and attaches it the base license of the product.

In  | HID | Hardware id
| BaseLID | Product license number
| AddonLID | Add-on license number
Ret | ReturnCode | Ok, error
| Message | Return message
| ActivationStatus | Ok, error
| AddonName | Name of the activated add-on

8.2.1.3 ActivateProduct
Activates the product.

In  | HID | Hardware id
| BaseLID | Product license number
Ret | ReturnCode | Ok, error
| Message | Return message
| ActivationStatus | Ok, error

8.2.2 RepositoryService
RepositoryService manages the distribution of updates that the user requests.
The Repository has the following function.

8.2.2.1 GetUserConfiguration
The current licenses and their version are passed in.
Then the return will be the licenses that are bound to the product, these are the licenses that the user should have. Also URLs to the next possible versions of the licenses will be returned if the user has an older version. This update can be in incremental steps. So the user might have to run this again after the update has been installed.

In  | HID | Hardware id
| BaseLID | Product license number
| LIDLIST | List with the current licenses and their version
Ret | ReturnCode | Ok, error
| Message | Return message
| CONFIGURATIONLIST | List of the licenses and the version that the user should have. If the user doesn’t have the version then it will also contain a list of URLs where an update can be downloaded from.
LIDLIST is a structure with the following items
LID        License number
Ver        The current version that is installed

CONFIGURATIONLIST is a structure with the following items
LID        License number
Ver        The latest version that the user can upgrade to or has
URLLIST    A list of URL where an update can be downloaded from

8.2.3 DataService
DataService manages and stores the user input. It receives information such as when the user starts the product, when the user ends the product. If the user tries to use a function that isn’t activated yet (used for sales pitch), the user can also send in question about the software.

Only one of the functions is displayed since the usage functions is similar to each other.

8.2.3.1 UpdateProductUsage
The product retrieves an offer that is presented to the user, if any.

In  HID            Hardware id
     BaseLID        Product license number
Ret ReturnCode    Ok, error
     Message        Return message
     GUID           GUID that is passed back when the product is terminating.

8.2.4 SalesService
SalesService manages and distribute sales pitches that are to be displayed to the user.

8.2.4.1 GetOffer
The product retrieves an offer that is presented to the user, if any.

In  HID            Hardware id
     BaseLID        Product license number
Ret ReturnCode    Ok, error
     Message        Return message
     URL            URL to the offer, if any
Database

In this chapter we will focus on the database model. Databases are an important part of products that are working with a lot of data and need to access and modify the data in a safe way. Inconsistent data must be avoided and speed is of the essence. In this Master’s thesis the database model is probably the most important element. It needs to store information about the software, the users, the licenses and the usage of the software.

The database will be divided and described in three different diagrams although they will be stored in the same database. These are Repository, Custom Data and License Usage.

9.1 Repository

We need to be able to store both complete (full) versions and updates in the database. But most of the times the user (in our case the product) will have to download the file from another server that is dedicated in distribution of files such as fileburst.com. This is to avoid the bandwidth problems that may occur for the users that needs to access license information that is stored in the same database.

The deployment repository (database) has a few similarities to the structure in the CCU model but has some changes. Version is included to be able to see what version the package is (this is different from the CCU model which included version on the components instead. This model has a slightly different approach and only stored a single binary as the product and/or update.) Type is included to distinguish if it is an update (delta file) or a complete release (full). This will let us store the both full versions and update version in the database or references to URLs. Distribution is added to the model; sometimes the distribution is both update and full whereas sometimes a distribution is only one of update or full. Binary is a replacement of the Software Package in the CCU model, the binary stores up to one binary file. This is the actual software that is to be distributed. DownloadURL contains URLs where the binary can be downloaded. These URLs can be the www.effectivestudios.com own page or distribution servers like fileburst.com.

All updates applies in successive order, this is to avoid complicated rules. The design was first built using a requirement list to make it flexible. In that way you could set up dependencies that require other updates or complete installations before the targeted one. It was built up with and, or. But it would be too error prone so to keep it clean and straight forward it was removed.

In Figure 9-1 we can see the table and their relations. The ProductInfo is in the same database but is described in another diagram.
The repository table (Table 9-1) contains packages.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK</td>
<td>Int</td>
</tr>
<tr>
<td>Name</td>
<td>Varchar(200)</td>
<td></td>
</tr>
</tbody>
</table>

Table 9-1 Table Repository

The packageList table contains the relationship between the package and the product. Table 9-2 shows the attributes, relations and data types. A product can be composed of several packages. This was a M:N relation that was solved by introducing this new table with the PKs from the tables package and productInfo.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PackageID</td>
<td>PK, FK, ref Package_ID</td>
<td>Int</td>
</tr>
<tr>
<td>ProductInfoID</td>
<td>PK, FK, ref ProductInfo_ID</td>
<td>Int</td>
</tr>
</tbody>
</table>

Table 9-2 Table PackageList

The package table stores software entities. It is an entry point for a product and can contain both updates and complete versions. As an example it can contain Site Studio 6 Pro Sv, Site Studio 6 Pro En, and Newsletter Studio 7 etc. The attributes, relations and data types are listed in Table 9-3. The productionStateTypeID is a FK that indicates the state that the package is in. The states
can be active, inactive or test. The idea is that if the packages are set to test then only request from in house IPs will have the packages recognized as active. This will let the test group perform test on the update (or complete version) before it is released to the customers. It is intended to be the final test. Of course it would be even better with a separate test environment with a complete replicated database. But with this the possibility for test can be used until such an environment exists. In the future it would be good to be able to restrict the update to only apply to a certain group of users. In this way the update can be distributed to a small set of customers to detect errors before all the customers receive the update.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK</td>
<td>Int</td>
</tr>
<tr>
<td>Name</td>
<td>Name of package</td>
<td>Varchar(200)</td>
</tr>
<tr>
<td>ProductionStateTypeID</td>
<td>FK, ref</td>
<td>Int</td>
</tr>
<tr>
<td></td>
<td>ProductionStateType_id, Indicates the state of the package.</td>
<td></td>
</tr>
</tbody>
</table>

**Table 9-3 Table Package**

The Version table in Table 9-4 contains all the version information that is bound to a package. Major and minor indicate how the production team rates the new release. The structure is Major.Minor. Usual a change in the major will result in a new package, i.e. going from Site Studio 6 to 7. Whereas minor is Site Studio 6.01 to 6.03.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PackageID</td>
<td>FK, ref Package_ID</td>
<td>Int</td>
</tr>
<tr>
<td>Major</td>
<td>Version number nnn.</td>
<td>Int</td>
</tr>
<tr>
<td>Minor</td>
<td>Version number .nn</td>
<td>Int</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the package</td>
<td>Varchar(500)</td>
</tr>
</tbody>
</table>

**Table 9-4 Table Version**

The licenseAgreement table (Table 9-5) contains the license agreement that is to be attached to a version. The intention of this is that when either a complete version or an update is retrieved then the product /server can display this text to the user before accepting the download request. This is plain text in this Master’s thesis. Another solution is to use a reference to a file location or store i.e. a html document as a blob to be able to format the text in an easy way.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VersionID</td>
<td>FK, ref Package_ID</td>
<td>Int</td>
</tr>
<tr>
<td>LicenseAgreement</td>
<td>License agreement text</td>
<td>Varchar(1500)</td>
</tr>
</tbody>
</table>

**Table 9-5 Table LicenseAgreement**

The table DistributionType contains information about the type of the distribution. It can be either an update or a full version. The update and full is included as an entry in the DistributionType table. The Distribution type table is shown in Table 9-6.
<table>
<thead>
<tr>
<th><strong>Attribute</strong></th>
<th><strong>Description</strong></th>
<th><strong>Data type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK, ID of the type</td>
<td>Int</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the type</td>
<td>Varchar(100)</td>
</tr>
</tbody>
</table>

Table 9-6 Table DistributionType

The Media, shown in Table 9-7.

<table>
<thead>
<tr>
<th><strong>Attribute</strong></th>
<th><strong>Description</strong></th>
<th><strong>Data type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK, ID of the type</td>
<td>Int</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the type</td>
<td>Varchar(100)</td>
</tr>
</tbody>
</table>

Table 9-7 Table Media

The table DownloadURL, shown in Table 9-8, stores information about where the resource can be located. MediaID is a FK to the Media table; this is to realize the inheritance in the model. In this case we used the horizontal mapping, it will be a bit slower to query, update, insert and delete than the vertical mapping but it makes it easier to change the attributes.

<table>
<thead>
<tr>
<th><strong>Attribute</strong></th>
<th><strong>Description</strong></th>
<th><strong>Data type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaID</td>
<td>FK, ref Media_ID</td>
<td>Int</td>
</tr>
<tr>
<td>URL</td>
<td>URL indicating where the resource can be located</td>
<td>Varchar(500)</td>
</tr>
<tr>
<td>MediaTypeID</td>
<td>FK, ID of which type of media this is. URL or Binary. URL in this case.</td>
<td>Int</td>
</tr>
</tbody>
</table>

Table 9-8 Table DownloadURL

The DownloadBinary table (Table 9-9) is similar to the DownloadBinary. The difference is that this table stores a binary chunk of data that can be retrieved.

<table>
<thead>
<tr>
<th><strong>Attribute</strong></th>
<th><strong>Description</strong></th>
<th><strong>Data type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaID</td>
<td>FK, ref Media_ID</td>
<td>Int</td>
</tr>
<tr>
<td>URL</td>
<td>URL indicating where the resource can be located</td>
<td>Varchar(500)</td>
</tr>
<tr>
<td>MediaTypeID</td>
<td>FK, ref MediaType_ID, which type of media this is. URL or Binary. Binary in this case.</td>
<td>Int</td>
</tr>
</tbody>
</table>

Table 9-9 Table DownloadBinary

In Table 9-10 we can see the table MediaType. It contains information about what type of media it is. The URL and Binary shown in the model (Figure 9-1 Repository structure) is a row in the MediaType table.


<table>
<thead>
<tr>
<th><strong>Attribute</strong></th>
<th><strong>Description</strong></th>
<th><strong>Data type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK, ID of the media type</td>
<td>Int</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the media type</td>
<td>Varchar(100)</td>
</tr>
</tbody>
</table>

Table 9-10 Table MediaType

The table ProductionStateType (shown in Table 9-11) is used to define three different types of activation status. Active, this is the state that indicates that all customers will be able to retrieve the item. Inactive, this is the state that indicates that no one will be able to retrieve the item. The last state is Test, this indicates that only requests made from the local IP will be able to retrieve the item. This is to make it possible to test updates and complete versions as if they were published live.

<table>
<thead>
<tr>
<th><strong>Attribute</strong></th>
<th><strong>Description</strong></th>
<th><strong>Data type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK, ID of the production state type</td>
<td>Int</td>
</tr>
<tr>
<td>Name</td>
<td>Name of production state</td>
<td>Varchar(100)</td>
</tr>
</tbody>
</table>

Table 9-11 ProductionStateType table

The table DistributionList (see Table 9-12 DistributionList table) forms a list of the different distributions that exists for a particular version. The DistributionList contains references to four tables, Version, ProductionStateType, DistributionType and Media. All these FK forms a primary key in this table. A distribution can either be a URL or a binary and an update or a complete version.

<table>
<thead>
<tr>
<th><strong>Attribute</strong></th>
<th><strong>Description</strong></th>
<th><strong>Data type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>VersionID</td>
<td>FK, ID ref Version_ID</td>
<td>Int</td>
</tr>
<tr>
<td>ProductionStateTypeID</td>
<td>FK, ID ref ProductionStateType_ID</td>
<td>Int</td>
</tr>
<tr>
<td>DistributionTypeID</td>
<td>FK, ID ref DistributionType_ID</td>
<td>Int</td>
</tr>
<tr>
<td>MediaID</td>
<td>FK, ID ref Media</td>
<td>Int</td>
</tr>
</tbody>
</table>

Table 9-12 DistributionList table

### 9.2 License Data

In Figure 9-2 we see the License Data structure. The License Data structure stores information about products, licenses and customers. As mentioned above the customers are not needed, but if a customer decides to register information about itself then that information is stored.

ProductInfo is information about a product. A product can be i.e. Site Studio 6 Pro En or Plus Package En. A productInfo can contain a list of other productInfos and thus be thought of as a bundle of products. The productInfo acts like a template, when a license needs to be created then the productInfo structure is used to create and link the licenses according to the template.

One example of a productInfo that contains two licenses is the Plus Package that contains extra concept templates and a support agreement, both valid for one year. Licenses can be of three different types; longterm, subscription and trial. Longterm has no end date, the license is forever. Subscription and trial version is basically the same; they both have the number of days
that the license is valid after the installation. Even though the subscription and trial license is similar they are separated since we need to be able to distinguish them from each other. This way we can show sales pitches for the trial user that is different than the subscription version. Each trial license number can be used by many users. The statistics about the users of the trial version is monitored via its HID and the valid days should be matched with the HID. A subscription type can be an additional package with concept templates that is paid once every year. Since licenses can shift from a customer to another they are tracked by the Owns table; this let us keep a history of the changes. If a license is misused the activation status can be set to Locked. The default value for this is inactive and when the user activates the license for the first time this is set to true.

![Figure 9-2 Basic License Data structure](image)

The table ActivationStatus (see Table 9-13 ActivationStatus table) is used to indicate status of a license. There are three different states; active, inactive and locked. When a license is added the usual state is inactive. When the user has a network connection then the server will receive information that the license is used and then it will be changed to active. To disable a license the state is set to locked. This can occur when a license has been used in concurrent. The three different states are added as rows in the ActivationStatus table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK, ID</td>
<td>Int</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the activation. Active, inactive and locked.</td>
<td>Varchar(100)</td>
</tr>
</tbody>
</table>

Table 9-13 ActivationStatus table

The customer table (Table 9-14) stores information that the customer has typed in about itself and then decided to register. This is not central to the Master’s thesis and all the specific user data is omitted here.
Ownership of a product can change over time. The Owns table (see Table 9-15) keeps a history of these changes. On side effect of the change of owner ship is that if we don’t store the old owner then the new owner will have the usage information attached to it. It will still be attached to the license but if we want to see who owned that license at that current time then we can use the owns table in the query to figure it out. When DateAdd and DateRemove is the same then the customer still owns the product.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK</td>
<td>Int</td>
</tr>
<tr>
<td>FirstName</td>
<td></td>
<td>Varchar(100)</td>
</tr>
<tr>
<td>LastName</td>
<td></td>
<td>Varchar(100)</td>
</tr>
<tr>
<td>…</td>
<td>… other user info …</td>
<td>…</td>
</tr>
</tbody>
</table>

Table 9-14 Customer table

The LicensType indicate how the license will be treated (Table 9-16). There are three different types of licenses. Longterm, these licenses never expire. Subscription, these licenses expires after n days. Trial, these licenses expires after n days too. The reason that they are separated is that for log reasons it is good to keep track of the licenses that is trial licenses and those that are subscription. The three different types is added as a row in the LicenseType table. The attribute days is moved to the LicenseType table as well. When it is a longterm license then the days will be set to 0.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK, ID</td>
<td>Int</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the license type. Longterm, Subscription and Trial.</td>
<td>Varchar(100)</td>
</tr>
<tr>
<td>Days</td>
<td>Number of days the license is valid. 0 if it is a longterm license.</td>
<td>Int</td>
</tr>
</tbody>
</table>

Table 9-16 LicenseType table

Since the company has products in several different languages a Lang table (Table 9-17) is used to store the languages. A product can then contain several languages. Here we have a M:N relationship that needs to be solved. By inserting an extra table named ProductLang the problem is solved.
As stated above, a product can be on several different languages. And also a product can be shipped containing several languages. To solve the M:N relationship a new table was introduced (Table 9-18). At the end of this chapter a redesigned database model is shown (Figure 9-5).

### Table 9-18 ProductLang table

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductID</td>
<td>FK, ref ProductInfo_ID</td>
<td>Int</td>
</tr>
<tr>
<td>LangID</td>
<td>FK, ref Lang_ID</td>
<td>Int</td>
</tr>
</tbody>
</table>

A product can be a bundle that consists of several sub products (Table 9-19). ProductID is the “base” product and ProductSubID is the sub products.

### Table 9-19 Includes table

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductID</td>
<td>FK, ref ProductInfo_ID</td>
<td>Int</td>
</tr>
<tr>
<td>ProductSubID</td>
<td>FK, ref ProductInfo_ID</td>
<td>Int</td>
</tr>
</tbody>
</table>

One goal for the company is to earn more money, of course. And one important aspect is to show the user what he/she can buy. To be able to accomplish that a Pitch table is created (Table 9-20). It is a list of all the sub products that we want to promote to the user. ProductID is the “base” product and ProductSubID is the sub products. The product can retrieve this list for all the licenses that is currently active and then combine it with the licenses that it has and thereby show a list of available updates to the user. This should be ok, since it will only pitch for items that require licenses that are already active. An alternative is to find a design that will make it possible to promote products that isn’t available without first buying a product in between.

### Table 9-20 Pitch table

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductID</td>
<td>FK, ref ProductInfo_ID</td>
<td>Int</td>
</tr>
<tr>
<td>ProductSubID</td>
<td>FK, ref ProductInfo_ID</td>
<td>Int</td>
</tr>
</tbody>
</table>

The ProductInfo (Table 9-21) is used as a template when creating licenses. With this information it will be possible to create and bind sublicenses. This part of the model could be developed further so it can be integrated with sales and generate licenses on demand when orders arrive and gets accepted. Besides the three different types of products (longterm, subscription and trial) a product is also a base product or not. Base product is products that can be bought with no concern of other products. They do not depend on others. Site Studio 6 Pro En is a base product while Plus Package is not since it needs to be attached to Site Studio 6.
### Table 9-21 ProductInfo table

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK</td>
<td>Int</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the product</td>
<td>Varchar(100)</td>
</tr>
<tr>
<td>LicenseTypeID</td>
<td>FK, ref LicenseType_ID</td>
<td>Int</td>
</tr>
<tr>
<td>LangID</td>
<td>FK, ref Lang_ID</td>
<td>Int</td>
</tr>
<tr>
<td>IsBaseProduct</td>
<td>Is it product that doesn’t need to be connected to any product</td>
<td>Bit</td>
</tr>
</tbody>
</table>

Licenses (Table 9-22) are the heart of the company. The company sells licenses both in boxed and online versions. To generate more profit add-ons will be a major focus to the company and an important aspect is to manage the licenses and keep track on what licenses are activated and not.

### Table 9-22 License table

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK</td>
<td>Int</td>
</tr>
<tr>
<td>LicenseNumber</td>
<td>License, i.e. SS06-NNNN-…</td>
<td>Varchar(100)</td>
</tr>
<tr>
<td>ProductID</td>
<td>FK, ref Product_ID</td>
<td>Int</td>
</tr>
<tr>
<td>ActivationStatusID</td>
<td>FK, ref ActivationStatus_ID</td>
<td>Int</td>
</tr>
</tbody>
</table>

### 9.3 License Usage

The License Usage (see Figure 9-3) contains information about the usage of the product. Each installation of the product creates the hardware signature (HID) which is passed to the server in the communication.

To the HID we have bound information about the usage. Version indicates what version of the software that is currently running, if the user updates/reinstalls this value is updated as well. Updates show a history of what updates that the user has installed. Configuration contains information about the user computer such as the OS. BugReport contains the information that the user has sent in as a bug report. ProductUsageData stores information about the products usage. This is information about installation, the running of the product when did the user start the product and when did it terminate, ProductUsageCT stores information about what templates the user has used.

Due to the HID in combination with the license we can keep track of one computer. If a new installation is made on a different computer a new HID is created. This let us monitor if a product is used by many different users and since we also store when the product is launched and when it is terminated we can also monitor if the product is used concurrent. If there is a specific computer that is using the software and we don’t want to lock the license then the HID for that computer can be locked.
Figure 9-3 License Usage structure

The table activation state is the same table as the one in the License Data diagram.

To track and store information about an individual installation a HID and the license number is used. The HID is a signature of the hardware on the user machine; in our case we generate it from the network card. We can set the activation status on individual installation and thereby i.e. lock a specific installation. However it would be more common to block the entire license instead. The HID table (Table 9-23) also contains the current version that the user are running on the particular installation as well as the configuration (operating system). These two elements are added as attributes in the HID table (in the redesigned database model at the end of this chapter those are removed). We could have used HID and LicenseID as key but to reduce the keys in the tables that are referring to this one we introduce a counter as a key. Still the HID and LicenseID are used when the License table query and the constraint is that these two has to be unique when combined.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>PK</td>
<td>Int</td>
</tr>
<tr>
<td>HID</td>
<td>Hardware signature</td>
<td>Varchar(50)</td>
</tr>
<tr>
<td>LicenseID</td>
<td>FK, ref License_ID</td>
<td>Int</td>
</tr>
<tr>
<td>ActivationStatusID</td>
<td>FK, ref ActivationStatus_ID</td>
<td>Int</td>
</tr>
</tbody>
</table>

Table 9-23 HID table

To store information about what updates that has been made on a particular installation we store it in the Update table (see Table 9-24). The relation to the HID table is made by using the
HIDID that is a counter in that table. The date indicates when the update was made. The major and minor version indicates what the version the update was.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIDID</td>
<td>FK, ref HID_ID</td>
<td>Int</td>
</tr>
<tr>
<td>Date</td>
<td>Installation date</td>
<td>Date</td>
</tr>
<tr>
<td>VersionMajor</td>
<td>Major version of the installed update</td>
<td>Int</td>
</tr>
<tr>
<td>VersionMinor</td>
<td>Minor version of the installed update</td>
<td>Int</td>
</tr>
</tbody>
</table>

Table 9-24 Update table

Several modifications are made to the database model (Figure 9-3). The Feedback and ProductUsageData table are removed. Instead a more flat structure is used with relations between directly to the HID table.

Bug report (Table 9-25) information is sent from the user. This will help us collect information about bugs or annoyances that the user has stumbled upon. The date and the version are also stored to better pin point the problem. The version is important since it can be an old version that the user is running. The date might tell us that a server was down that period of time and thus not a bug.

Both the bug report and the usage question might be disabled since the company wants to build a portal on the web where the user will perform these tasks instead.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIDID</td>
<td>FK, ref HID_ID</td>
<td>Int</td>
</tr>
<tr>
<td>Date</td>
<td>PK, Report date</td>
<td>Date</td>
</tr>
<tr>
<td>VersionMajor</td>
<td>Version running</td>
<td>Int</td>
</tr>
<tr>
<td>VersionMinor</td>
<td>Version running</td>
<td>Int</td>
</tr>
</tbody>
</table>

Table 9-25 BugReport table

The user can send questions about the software (Table 9-26). This can i.e. be questions of how to solve a particular problem or information about add-ons.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIDID</td>
<td>FK, ref HID_ID</td>
<td>Int</td>
</tr>
<tr>
<td>Date</td>
<td>Question date</td>
<td>Date</td>
</tr>
<tr>
<td>Question</td>
<td>Question asked</td>
<td>Varchar(2000)</td>
</tr>
<tr>
<td>VersionMajor</td>
<td>Version running</td>
<td>Int</td>
</tr>
<tr>
<td>VersionMinor</td>
<td>Version running</td>
<td>Int</td>
</tr>
</tbody>
</table>

Table 9-26 UsageQuestion table
If the product is about the crash or encounters an error (Table 9-27) that it can recover from information about this is sent to the server and stored. This will let us track down errors and see if they are common.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIDID</td>
<td>FK, ref HID_ID</td>
<td>Int</td>
</tr>
<tr>
<td>Date</td>
<td>Error date</td>
<td>Date</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>Error reported</td>
<td>Varchar(2000)</td>
</tr>
<tr>
<td>VersionMajor</td>
<td>Version running</td>
<td>Int</td>
</tr>
<tr>
<td>VersionMinor</td>
<td>Version running</td>
<td>Int</td>
</tr>
</tbody>
</table>

Table 9-27 ErrorReport table

Information about the start and end of the product is stored (Table 9-28) to be able to calculate statistics about how many users that are running the software (overall). The information is also used to see if a license is used by multiple users concurrent. In some cases it is ok since there could be a product with 10 licenses. To be able to measure the concurrent usage a start date is stored; the end date is set to the start date plus 10 minutes. This will give us a time span even if the user doesn’t send a close connection response which could occur if it crashes. The response from the server contains a GUID that was generated on the server when the information was stored. The GUID is also stored in the table. This is to be able to match a close connection call from the product with the right row. The user can start multiple instances off the product on the same computer. From the information that is stored we can look at all the HIDs that are bound to a license and compare the time intervals to see if there are any concurrent uses. When the product sends a response that it is being terminated the information about the DateEnd and SentConnectionClosed is updated. The response contains the GUID that was returned when starting the product.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIDID</td>
<td>FK, ref HID_ID</td>
<td>Int</td>
</tr>
<tr>
<td>DateStart</td>
<td>PK, When the product was launched</td>
<td>Date</td>
</tr>
<tr>
<td>DateEnd</td>
<td>When the product terminated</td>
<td>Date</td>
</tr>
<tr>
<td>IP</td>
<td>The IP number of the user</td>
<td>Varchar(15)</td>
</tr>
<tr>
<td>SentConnectionClosed</td>
<td>If the user has sent a closed connection response</td>
<td>Bit</td>
</tr>
<tr>
<td>VersionMajor</td>
<td>Version running</td>
<td>Int</td>
</tr>
<tr>
<td>VersionMinor</td>
<td>Version running</td>
<td>Int</td>
</tr>
<tr>
<td>ConnectionGUID</td>
<td>GUID that the user will return when the connection ends.</td>
<td>UniqueIdentifier</td>
</tr>
</tbody>
</table>

Table 9-28 ProductUsageConnection table

When the user uses a custom template (CT) from the online resources feedback about this is sent to the server. This will let us see what templates are the most popular ones and it might indicate what template that needs to be modified (Table 9-29).
9.4 Updated database model

During the sections above we have performed some modification of the database models. Below are the updated models (Figure 9-4, Figure 9-5 and Figure 9-6).

9.4.1 Repository

Figure 9-4 Updated Repository
9.4.2 License Data

Figure 9-5 Updated License Data

9.4.3 License Usage

Figure 9-6 Updated License Usage
9.5 Summary

We have now designed the database that is needed to store all information needed. The database consisted of three logical parts; Repository, License Data and License Usage. The repository is used to store the distributions; the License Data is used to store information about the product, licenses, customers and their status. The last part is License Usage that stores information about the usage of the product.
10 Summary

This is an overview of what was achieved in this Master’s thesis.

Figure 10-1 shows the overview of the final structure. SSL was added to secure communication, the Server has been divided to four web services and the database consists of three logical parts.

10.1 Client

A Software licensing system module was designed. This module communicates with the web services and should be included in the product. If the product is running in trial mode then the pattern Time-based is used to validate the time. Otherwise the validation occurs on the server.

10.2 Server

On the server side four web services was created.

- LicenseService that performs the operations regarding licenses.
- UpdateService that performs the operation of distributing the information needed to update the user’s product to the latest version. The updates are returned as URLs where the updates can be downloaded. The URLs can refer to external servers to avoid bandwidth problem at the company.
- DataService that stores information about the usage of a product.
- SalesService that retrieves a customized offer that should be presented to the user, if any.

10.3 Database

The database consists of three different logical parts that are stored in the same database.

- The Repository stores information about the product. Such as the version, where it can be downloaded etc.
- License Data stores information about the license, the product and the customer. The license is the core of the system. It is the license and the repository that forms the user’s configuration. The updates that are distributed should be digitally signed so that the client can validate that the files hasn’t been tampered with.
- License Usage stores information about the usage of the product. By using the hardware signature the information about a license is locked to a particular computer.
10.4 Communication

To secure the communication SSL is used. To implement SSL should be a minor issue since SSL certificates can be bought and installed on the server.
Conclusions

Online software distribution system stores the user’s configuration online, this turns out to be quite powerful since it will let the user install a base version on a new computer then enter a license key and all the missing parts will be downloaded and installed on the new machine. Also if an add-on is bought then the files that are needed for that add-on will be downloaded and installed in the synchronization step.

When decomposing the problem of designing an online software distribution system you’ll find that the actual corner stone in an online software configuration comes down to designing a well structured database.

11.1 Goals

The Master’s thesis had goals defined at the beginning; these are the goals and the solution to them.

11.1.1 Zero administration

The activation and upgrade is autonomous and therefore requires zero administration. This is since the user buys a license online on an external web page. The serial number is then entered inside of the product. A request is sent to the server with the base license (product license) and the license of the add-on (in case of product activation). The server receives the request and binds the add-on license to the base license. The product then synchronizes with the server by sending a request for the new configuration. The request contains the current licenses and their version and the server processes this list and returns a list of the licenses that should be installed and their next version. The version returned could be the final version or a step in the update process since an update could be in several steps, i.e. 1.3->1.4->1.5. The update is pointed out by an URL. The synchronization step could be performed several times since some updates require a restart of the product.

11.1.2 Online validation

The validation of licenses is performed at the server thereby adding more value to owning a legal license since a product needs to be valid or the latest upgrades won’t be installed. Actually all requests to the web services are validated since the web services are stateless. All the information needed to perform an operation on the web service is sent in the request and a validation of the license is made at the start of the web services operations.

11.1.3 Customize offers for trials

The user running a trial version will retrieve a customized offer from the server but the offer is not based on the user’s statistics of using the product. The customized offer is made in the database (in the license data).

11.1.4 Update trial into full version software

An installation with a license that is missing parts downloads the missing parts when the synchronization with the server occurs. This makes it possible for the user to update a trial version to a full version in an automatic process.

11.1.5 Distribute upgrades

By adding the updates to the Repository the user’s will retrieve the new version on the next update.
11.1.6 Disable expired upgrades
The expiration of add-ons is monitored online since the user’s configuration is stored on the server.

11.2 Negative aspects
With a distribution system like this there could be some negative aspects as well.

11.2.1 Increasing demand
The demands from the customer regarding updates could increase. The demand for update can be to fix a problem as well as the psychological aspect that an update should occur often to show that the product is evolving.

11.2.2 Error in upgrade
Software distribution is very powerful. The distribution of an upgrade that contains error could be spread very fast too many customers. This could have a very negative impact on the customers’ computers and trust.

11.3 Future work
There is much functionality that can be integrated in a distribution system like this. These are only two of many interesting aspects.

11.3.1 Multiuser license
With a system like this it is also possible to keep track of multiuser licenses even though this rule can’t be strict since a user can have the product installed on more than one computer. Instead it can be complemented with the ability to track concurrent uses of a license using the start and end time of the product.

11.3.2 Online business
Online business has a great potential since it can collect data in real time. With a system like this it is possible to monitor all kinds of statistics about the user. This can then be used to send Newsletters or display an offer when the product starts up that is based on the user’s profile.

11.3.3 Validate upgrade
As stated in the negative aspects an upgrade can have a negative impact if it contains errors. One way of limiting the problem is to make it possible to only distribute the upgrade to a few selected customers, i.e. 100 customers. If there are no reports of errors then more customers can retrieve the update.


Appendix A

A.1 Scenarios

<table>
<thead>
<tr>
<th>Scenario name</th>
<th>Use on-line resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participating actor</td>
<td>Product</td>
</tr>
<tr>
<td>Flow of events</td>
<td>1. The user has put views a list of resources, in this case concept templates. The user selects one of the templates. The product receives request and contacts the server about the request.</td>
</tr>
<tr>
<td></td>
<td>2. The server receives the request for the usage of the resource. The license retrieved in the call is looked up and all the licenses bound to that license is also retrieved. A validation that the user should have access to the resource is made. If it is ok then the resource is returned to the product.</td>
</tr>
<tr>
<td></td>
<td>3. The product receives the resource</td>
</tr>
</tbody>
</table>

Table A-1 Scenario Use on-line resource.

<table>
<thead>
<tr>
<th>Scenario name</th>
<th>Evaluate product (no updates)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participating actor</td>
<td>Product</td>
</tr>
<tr>
<td>Flow of events</td>
<td>1. The product is started in trial mode.</td>
</tr>
<tr>
<td></td>
<td>2. The product calls the Software licensing system to check if the license is valid.</td>
</tr>
<tr>
<td></td>
<td>3. The Software licensing system validates the license, that is it checks if the time limit has been exceeded or not. If it has exceeded then the product will retrieve a response that it should end. Otherwise it will retrieve a response that it is ok to continue.</td>
</tr>
<tr>
<td></td>
<td>4. The product then checks if the configuration on the user’s machine is ok by calling the Software licensing system.</td>
</tr>
<tr>
<td></td>
<td>5. The Software licensing system receives the request and connects to the server requesting the next state of the user’s configuration if any. In the request the current version of the installed components is also passed in.</td>
</tr>
<tr>
<td></td>
<td>6. The server checks if there any of the version has an updated version. If so the newest possible version is added to the response. It could be incremental updates. It also adds the licenses that wasn’t in the request (new ones) and removes licenses from the request that shouldn’t be in the configuration. All returned licenses also include the state and if any is a subscription then the end date is also added to the response.</td>
</tr>
<tr>
<td></td>
<td>7. The Software licensing system receives the response and validates if any updates is needed. A list of the updates that is needed is passed back the product with the new version number and the location of the file.</td>
</tr>
<tr>
<td></td>
<td>8. If no updates then the product calls the Software licensing system to</td>
</tr>
</tbody>
</table>
retrieve an offer.

9. The Software licensing system calls the server requesting a offer for the license.

10. The Server checks if there is an offer for the license and in that case returns the offer.

11. The Software licensing system return the offer to the product.

12. If an offer was received then the offer is presented to the user otherwise a default offer is presented.

13. The product is not running.

<table>
<thead>
<tr>
<th>Table A-2 Scenario Evaluate product.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario name</td>
</tr>
<tr>
<td>Participating actor</td>
</tr>
<tr>
<td>Flow of events</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

| Table A-3 Scenario Activate add-on. |

| Use case name | Get offer |
| Participating actor | product |
| Flow of events | 1. The product is running in trial mode and tries to get an offer to display to the user. A request is sent to the Software licensing system for a new offer that is to be displayed to the user. |
| | 2. The Software licensing system sends a request to the Server with the base license (product license) and the hardware id asking for an offer to display to the user. If no connection can be made then a message is returned that no offer currently exists. |
| | 3. The Server receives the request and checks in the persistent media if there is an offer for the requested license. If found it is return otherwise a message that no offers exists is returned. |
4. The Software licensing system checks the response and if there’s an offer returned then that offer is returned to the product. Otherwise a message that no offers exists is returned to the product.

5. If the response contains a new offer then that offer is displayed to the user, otherwise the product will retrieve the default offer and display that instead.

Table A-4 Use case, Get an offer.

<table>
<thead>
<tr>
<th>Use case name</th>
<th>Activate add-on</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participating actor</td>
<td>Product</td>
</tr>
<tr>
<td>Flow of events</td>
<td>1. The product is running in full mode the user has entered a license id to activate an add-on. A request is sent to the Software licensing system with the base license and the license to activate.</td>
</tr>
<tr>
<td></td>
<td>2. The Software licensing system sends a request to the Server with the base license (product license), the license of the add-on to activate and the hardware signature asking to register the add-on.</td>
</tr>
<tr>
<td></td>
<td>3. The Server receives the request and checks in the persistent media if the base license and the add-on license are valid. If it is valid then the user’s configuration on the server is updated to include the new add-on license. And return an ok to the Software licensing system.</td>
</tr>
<tr>
<td></td>
<td>4. The Software licensing system returns the response to the product.</td>
</tr>
<tr>
<td></td>
<td>5. If the response is ok that the odd-on was registered the user gets information that the product needs to be restarted (the Automatic update of user’s configuration on user’s machine use case will be executed). If it couldn’t be registered the user receives information about it.</td>
</tr>
</tbody>
</table>

Table A-5 Use case, activate add-on

<table>
<thead>
<tr>
<th>Use case name</th>
<th>Use on-line resource, A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participating actor</td>
<td>Product</td>
</tr>
<tr>
<td>Flow of events</td>
<td>1. The product is running in trial or full mode and needs to access on-line resources. A request is sent to the Software licensing system to get the hardware signature.</td>
</tr>
<tr>
<td></td>
<td>2. The Software licensing system sends a request to the Server with the base license (product license) and the hardware id requesting the on-line resource A.</td>
</tr>
<tr>
<td></td>
<td>3. The product sends a request to the Server with the base license (product license) and the hardware id requesting the on-line resource A.</td>
</tr>
<tr>
<td></td>
<td>4. The Server receives the request and validates that the</td>
</tr>
</tbody>
</table>
license should have access to the resource. If ok, then the resource A is sent back. Otherwise an error message is sent back.

5. If the response contains a resource A then it is downloaded otherwise a message is displayed to the user.

<table>
<thead>
<tr>
<th>Table A-6 Use case, use on-line resource, A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case name</td>
</tr>
<tr>
<td>Participating actor</td>
</tr>
<tr>
<td>Flow of events</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

| Table A-7 Use case. Send product feedback (start product). |
## Appendix B

### Entity object

<table>
<thead>
<tr>
<th>Product (PID, Licensenumber)</th>
<th>Product that contains a PID that identifies the product and a Licensenumber that identifies a unique (if not a trial version) instance of the product.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HID</td>
<td>Key that identifies a computer.</td>
</tr>
<tr>
<td>User</td>
<td>User owning the product.</td>
</tr>
<tr>
<td>Message</td>
<td>Message containing text</td>
</tr>
<tr>
<td>LicenseList</td>
<td>List that contains several licenses</td>
</tr>
<tr>
<td>SalesPitch</td>
<td>An offer that is presented to the user</td>
</tr>
<tr>
<td>UpdateList</td>
<td>A list of updates that can be installed</td>
</tr>
<tr>
<td>BugReport</td>
<td>Information about an error that the user sends</td>
</tr>
<tr>
<td>ErrorReport</td>
<td>Information about an error that the product has generated</td>
</tr>
<tr>
<td>UsageQuestion</td>
<td>Question regarding the product sent by the user</td>
</tr>
</tbody>
</table>

**Table B-1 Entity objects**

### Control object

<table>
<thead>
<tr>
<th>ValidateLicenseControl</th>
<th>Manages the ValidateLicense function on the client. This object is created when the client needs to validate the license.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateControl</td>
<td>Manages the Update off the client. This object is created when the client needs to retrieve a list of the version of some components and where to download the version in case of an update is needed. The list is retrieved from the server.</td>
</tr>
<tr>
<td>OfferControl</td>
<td>Manages the process of retrieving updates for a user. This object is created when the client is updating the commercial banners that will be displayed to the user during the start of the product.</td>
</tr>
<tr>
<td>ActivateAddonControl</td>
<td>Manages the Activation of add-on on the client. This object is created when the client tries to activate an add-on. The licenses are then activated on the server.</td>
</tr>
<tr>
<td>UpdateProductUsageControl</td>
<td>Manages the UpdateProductUsage function on the client. This object is created when the client need to send information about the product usage to the server. The information is then stored on the server.</td>
</tr>
</tbody>
</table>
UpdateProductUsageFunction RequestControl
Manages the UpdateProductUsageFunction function on the client. This object is created when the client needs to send information about the usage of a function that is not activated yet to the server. The information is then stored on the server.

UpdateBugReportControl
Manages the UpdateBugReport function on the client. This object is created when the user wishes to send information about a bug to the server. The information is then stored on the server.

UpdateUsageQuestionControl
Manages the UpdateUsageQuestion function on the client. This object is created when the user wishes to send information about a bug to the server. This information is then stored on the server.

UpdateErrorReportControl
Manages the UpdateErrorReport function on the client. This object is created when an error occurs on the client and trapped by the product. This information is sent to the server and stored there.

Table B-2 Control objects

### B.1 Sequence Diagrams

Figure B-1 shows the sequence diagram for the Send product feedback (start product). The product sends this message when the product is started. The server stores an entry in the persistent media and returns a GUID that the product passes with the call when the product terminates.

![Sequence Diagram](image)

**Figure B-1 Send product feedback (start product).**
Appendix C

Figure C-1 CCU - Delivery process data model
Figure C-2 CCU - Activity process data diagram