Aalto University
School of Science and Technology
Faculty of Information and Natural Sciences
Degree programme of Computer Science and Engineering

Michael Wikberg

Software license management from system-integrator viewpoint

Master's Thesis
Espoo, April 30, 2010

Printing date: June 3, 2010

Supervisor: Professor Tuomas Aura, Aalto University
Instructor: Jörgen Westerling M.Sc. (Tech.), eCraft Oy Ab

Aalto University
School of Science and Technology
Faculty of Information and Natural Sciences
Degree Programme of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Michael Wikberg
Title of thesis:
Software license management from system-integrator viewpoint

Date:	April 30, 2010	**Pages:** 14 + 55
Professorship:	Data Communications Software	**Code:** T-110
Supervisor:	Professor Tuomas Aura	
Instructor:	Jörgen Westerling M.Sc. (Tech.)	

Selling software and its components as a service for use by a widely distributed user base requires some effort on managing the licenses. This thesis is a study on how a license management system for a small or medium sized Independent Software Vendor could be built, by identifying the requirements and potential problems.

The framework requirements were identified while planning future software and features, the distribution of software and new invoicing possibilities as part of my daily work. Current licensing literature and best practices were studied to get an overview of license management technology, and this information was then used as a basis for the proposed architectural design.

The results consist of an analysis of identified actors and their roles, a plan for creating a license management system and finally an overview of security issues and how they are taken into consideration. The plan includes user interfaces, database schemas for license tracking, backend programming interfaces and some thoughts on integration to the existing invoicing system.

Keywords: software licensing, license management, SaaS, ISV, licensing framework
Language: English

Aalto-universitetet
Tekniska högskolan

Aalto-universitetet
Tekniska högskolan
Fakulteten för informations- och naturvetenskaper
Utbildningsprogrammet för datateknik

SAMMANDRAG AV
DIPLOMARBETET

Utfört av: Michael Wikberg

Arbetets namn:
Software license management from system-integrator viewpoint

Datum:	Den 30 April 2010	**Sidantal:** 14 + 55
Professur:	Datakommunikationsprogram	**Kod:** T-110
Övervakare:	Professor Tuomas Aura	
Handledare:	Diplomingenjör Jörgen Westerling	

Licenshantering är en viktig fråga vid försäljning av mjukvara och dess komponenter som tjänster. Användarna och deras behov varierar enormt, vilket ställer stora krav på systemet. Detta diplomarbete är en studie i hur ett licenshanteringssystem för ett litet eller medelstort mjukvaruföretag kunde byggas upp, genom att identifiera kraven och potentiella problem.

Systemkraven samlades in vid planering av framtida mjukvara och andra lösningar, som en del av mitt dagliga jobb. För att få en överblick av de bästa metoderna för implementering av detta sorts system gjorde jag en litteraturstudie där motsvarande och relaterade lösningar undersöktes.

Resultaten består av en analys av intressenterna och deras roller, en plan för implementeringen samt en genomgång av säkerhetsaspekter och hur de har beaktats. Planen innehåller användargränssnitt, databasscheman, programmeringsgränssnitt och tankar kring integration mot det existerande faktureringssystemet.

Nyckelord:	programvarulicensiering, licenshantering, SaaS, ISV licensieringsramverk
Språk:	Engelska

Acknowledgements

I would like to thank my employer eCraft Oy Ab, especially my bosses, for providing the thesis subject and allowing me to write during office hours.

A big thank you also goes to my sister, Milla, who did the proofreading and helped me get rid of other mistakes too.

Last but not least, I would like to thank my friends and family for supporting and encouraging me throughout the writing process.

Espoo, April 30th 2010

Michael Wikberg

Abbreviations and Acronyms

.NET	Microsoft .NET Framework
Cloud	Cloud computing is a way to share computing resources like CPU and storage over a network
API	Application Programming Interface
CA	Certificate Authority
CRL	Certificate Revocation List
CRM	Customer Relationship Management
ERP	Enterprise Resource Planning
IP	Intellectual Property
HTTP(S)	Hypertext Transfer Protocol (Secure)
ISV	Independent Software Vendor
PKI	Public Key Infrastructure
SaaS	Software as a Service
SDK	Software Development Kit
SQL	Structured Query Language
URI	Universal Resource Identifier
VSTA	Visual Studio Tools for Applications
VSTO	Visual Studio Tools for Office
XACML	eXtensible Access Control Markup Language

Contents

List of Tables

List of Figures

List of Listings

Chapter 1

Introduction

While planning new software that is to be sold on a service basis, the need for a framework for handling licenses, user authorizations and software configurations emerges. The framework should be usable for both existing and future developed software applications and modules. The software vendor wants to sell licenses, both for software as a service (SaaS) and other types of applications. Where possible, the customers want to get a ready to use, prepackaged solution, and for this they are willing to pay license fees. This approach is appealing in contrast to the conventional way of working closely with the vendor to create custom software solutions or to customize existing software, where the front-up costs are large.

The framework solution has to be very generic, so that it can be used for any new software being developed (and where applicable, for already existing software). Sample applications would include web sites, .NET applications (both online and offline), mobile device software (both for smartphones and more specialized devices), software components and hosted applications. Licenses could be invoiced by the time the license or application has been active, by usage count or by other rules where needed.

For applications caching or storing sensitive data, like confidential customer information, sales figures, manufacturing details etc., the stored data should be unusable for anyone after the license expires. This is, among other things, to prevent unauthorized sharing of the data with some other party, like a competitor, once the user's contract has expired.

This thesis focuses on finding working solutions and current best practices for implementing a software license management framework and its components in a portable and secure way. First the stakeholders are identified and their requirements listed through sample scenarios. A literature study is then

performed to find the current best practices for implementing the features and an architecture for the framework is proposed. Finally, security concerns, implications and their solutions are presented.

Chapter 2

Background

This chapter lists some sample scenarios, the involved parties (who will come in contact with the licensing framework), and then enumerates their interests. To be able to create a working licensing model and infrastructure, it is important to understand the differing needs and expectations of these parties. Finally, the combined system requirements are presented.

2.1 Scenarios

These scenarios are slightly modified actual customer projects, slightly altered to anonymize the customer and to simplify some non-relevant details.

2.1.1 Scenario A

Customer A manufactures large machinery and also services the equipment. They have a backend system that handles all their manufactured machinery, their component structure and technical data. This data is administered through a custom application, which is used by both employees of Customer A and their subcontractors. The subcontractors can only see relevant parts of the customer provided data, and naturally their own data, and manage structure and drawing revisions of these. The customer employees again are mostly responsible for consolidating the data from the subcontractors, and to make sure the designs are valid and complete (including user manuals, price lists and change history).

Retailers of A can access product information through a web shop (which uses data from the backend) and place orders on behalf of their customers

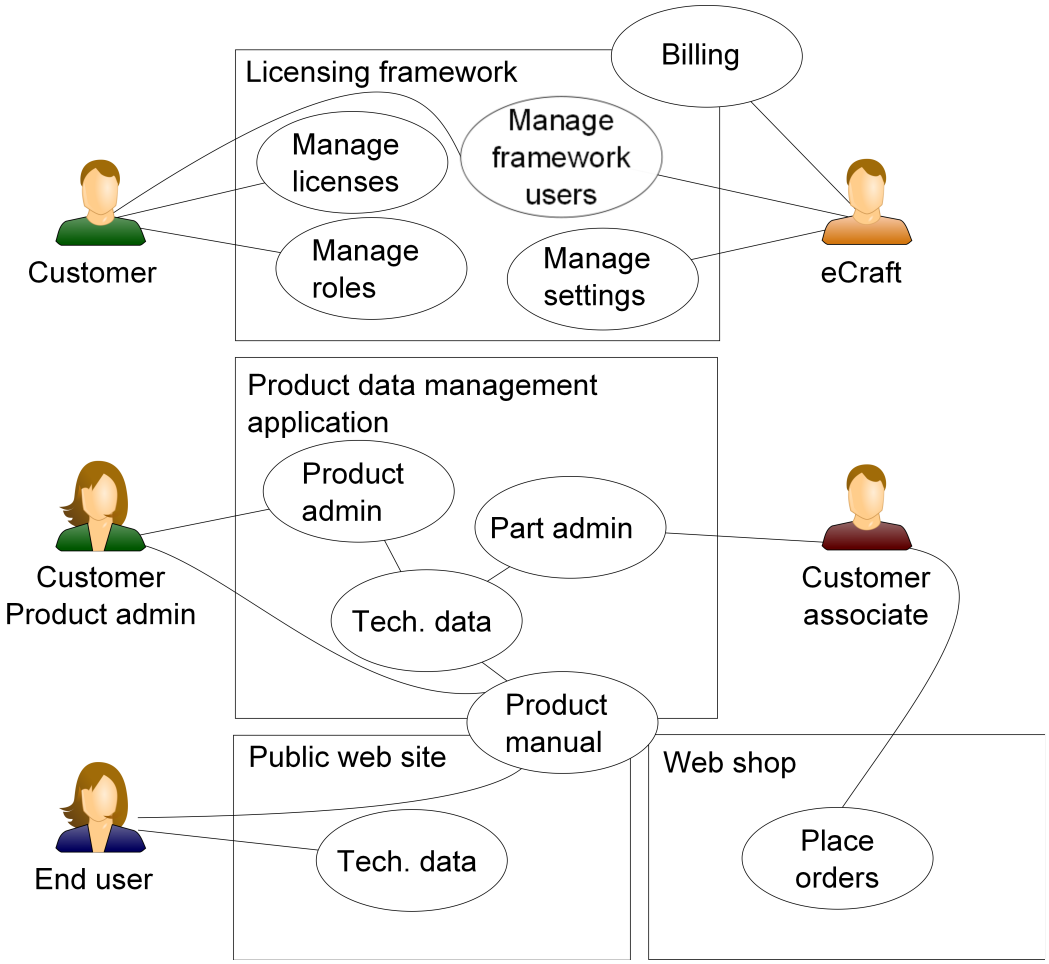

Figure 2.1: Scenario A

(end users for this purpose).

End users can access some details about their specific equipment through a public portal. These details include performed maintenance and upcoming planned service details.

Product administrators and subcontractors have completely different roles, but all required features are implemented in the same smart client software, which requires a license to work. The backend is also licensed, as is the use of the web shop. Data published for the end users falls under a single "publication" license.

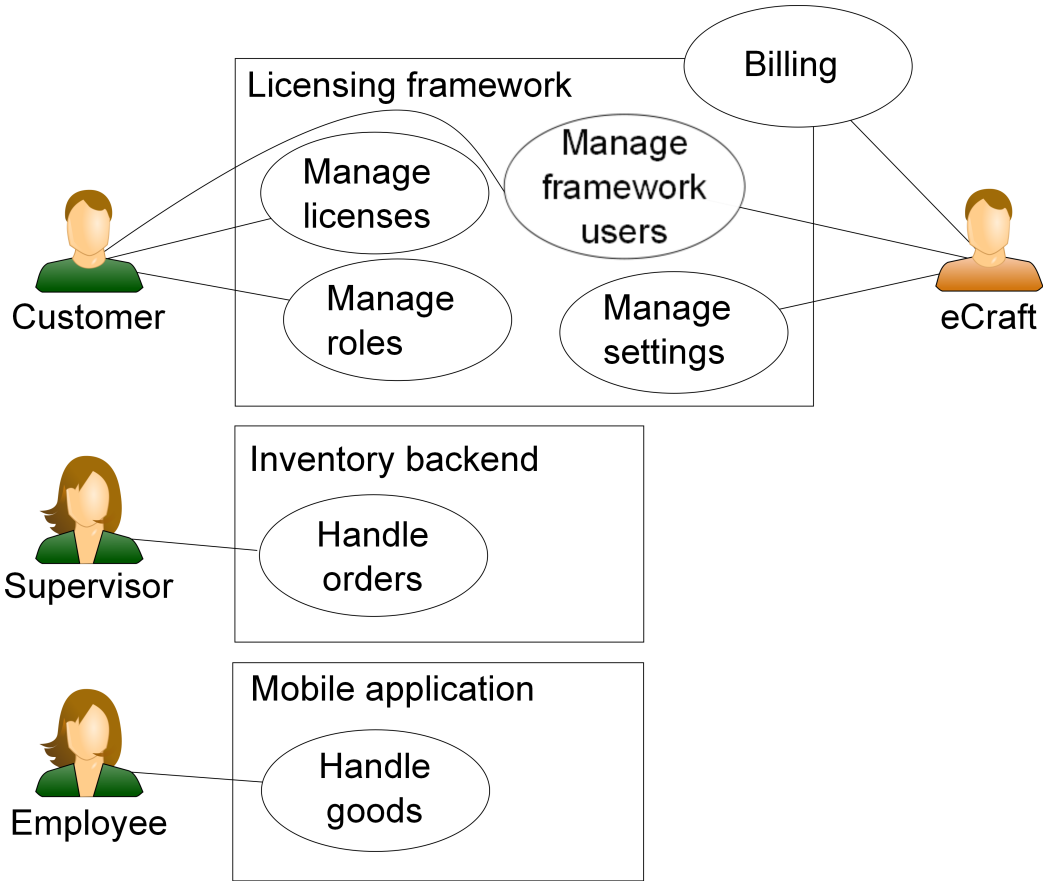

Figure 2.2: Scenario B

2.1.2 Scenario B

Customer B, being in the wholesale business, has several large warehouses and a customized inventory system. To keep track of where the goods are stored, they utilize mobile devices which run a custom inventory frontend on Windows Mobile and interact with the inventory system through a wireless network (or by synchronizing data in batches through a cable, when needed). The inventory backend system has one license, and each mobile device needs its own license.

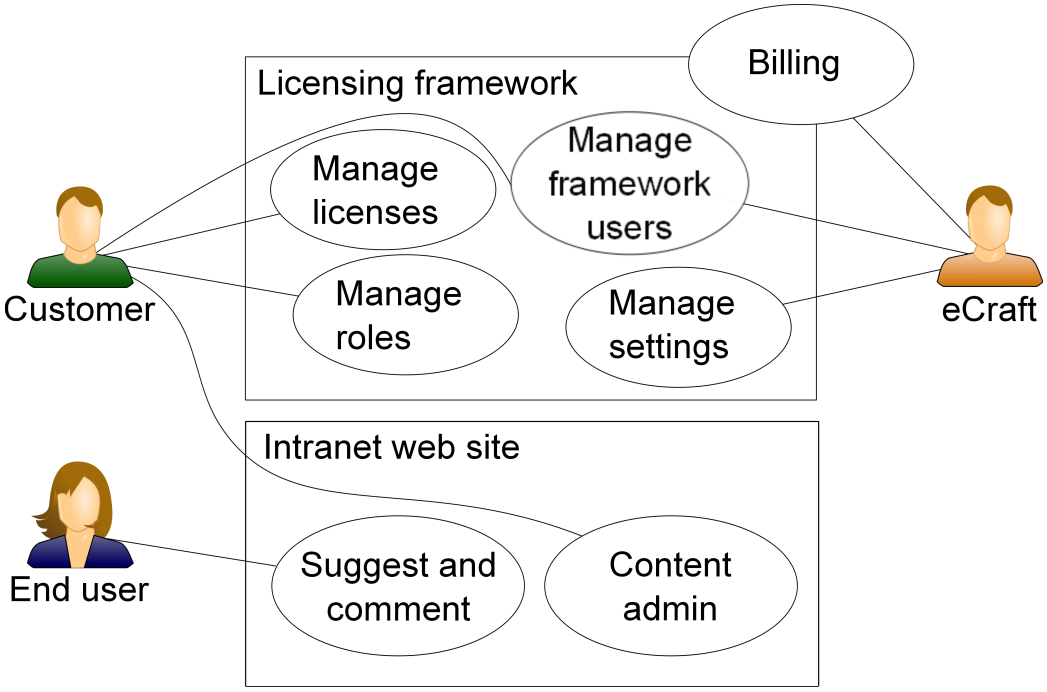

Figure 2.3: Scenario C

2.1.3 Scenario C

Customer C has a web application, where users can enter and comment on work related improvement proposals, on their intranet. The application is licensed to the whole company, but there are a few administrative users with elevated privileges. The management is interested in following up on user activity (mainly logon counts), which naturally is possible to implement separately, but is included in the licensing and role management system they will get anyway.

2.2 Stakeholders

This section introduces the stakeholders who will come in contact with the licensing framework, either directly or indirectly. The stakeholders are identified through the use of the scenarios mentioned in Section 2.1, but a generalization for most customer related projects can be made.

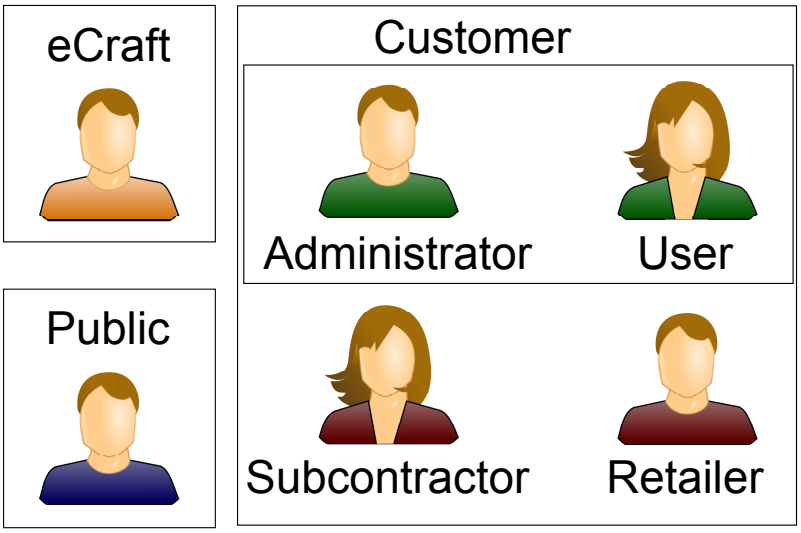

Figure 2.4: Stakeholders

2.2.1 eCraft Oy Ab

eCraft Oy Ab (also referred to as eCraft in this thesis) is a small Finnish company, who specializes in custom and customized software and integrations for the manufacturing industry. The most obvious part of eCraft that benefit from the system is probably the accounting department. Invoicing is made easier when the system automatically provides all needed information, and only the correctness of the content needs to be verified. The manual work of figuring out how many licenses are used and updating of their status (for new sales and revoked licenses) is mostly done by the customers.

Also sales personnel, who can use the framework features such as sales material, benefit. The most important and easy to understand features are full licensing control for the customer, fast response time for changes, cost prediction and real-time usage reports.

Finally there is the biggest group of people, the developers and architects, who can use the framework API to add licensing and configuration features to their code. Traditionally, these kinds of features have been developed separately for each piece of software, which results in lots of unnecessary overhead.

2.2.2 Customer

The customers, considered in this thesis, are almost all current eCraft customer companies and naturally also future ones. Both small and big companies might want to buy future software or migrate already existing solutions to a new licensing scheme.

It would be hard to use the framework as a sales argument, unless there are some real benefits for the customer. Depending on the type of software, there is usually a project or product manager who is responsible for the data handled by the software, and who possibly also allocates licenses and roles for the users. The framework needs to support several levels of management-type licenses, and an arbitrary amount of access rights (authorizations and roles), as defined by the software requirements in each individual case.

2.2.3 Customer subcontractor

In the manufacturing industry, which is the primary customer segment for eCraft, there are usually subcontractors who manufacture some parts of a larger machinery or equipment. There are usually technical writers who commit specifications and drawings to the customer.

2.2.4 Customer retailer

Sales representatives use software to retrieve up to date product information, place orders etc. This group needs offline capabilities, since it's not possible to have a working Internet connection in all places.

Service personnel and others need technical data, spare part information and service history for the equipment. This group cannot do their job if the data is unavailable.

2.2.5 Public or end users

There can also be "end user" types of users, who usually don't care much about what software they are using, on which conditions or how it actually works; it should just work. The licensing needs to be as transparent, preferably invisible, and easy to use as possible.

End users can access publicly available and non-confidential data, these users would probably not have a personal license, but instead the information is

made available by the customer through a publication license. In some cases, the user could access information about their specific equipment (like service history, upcoming services, etc.).

2.3 Stakeholder interests

For the licensing framework to be useful, it has to take the interests of all involved parties into account. This section presents some of the features required by stakeholders.

2.3.1 eCraft

Having a common framework for handling customer licenses and software specific configurations and roles is getting more important, especially when selling prepackaged software as a complete solution. It should be fast and simple to set up the environment for a new customer, both because the customer wants the service to be ready for use as soon as possible after the buying decision, and because the vendor wants to minimize the overhead involved.

The licensing system needs to be transparent and easy to explain, while still providing adequate protection for the customer data. Not all risks can be completely avoided, but in the end, it is up to the customer to decide what is acceptable and what is not.

Since the customers are managing their licenses themselves, it is up to the software vendor to make sure that the customer understands the implications. eCraft cannot be held responsible for any problems regarding licenses as a result of direct actions by the customer license administrators and data access through the use of compromised but not inactivated licenses.

Auditing data can be useful when the pricing level of licenses is not yet decided. The license pricing will probably depend on how much value the vendor expects the customer to receive through use of the software. This implies that billing might not be directly related to the amount of users, but rather to perceived customer advantage and the types of users. Some users enjoy a large benefit from using the software - these licenses could be more costly. Other users get some benefit, while some might not benefit very much at all. The latter group will probably not be interested in paying license fees at all. Exact pricing methods and levels are up to the sales force and management to decide.

2.3.2 Customer

The customer's main goal is to quickly acquire access to, and manage authorizations for services provided by eCraft. The decision to try out new software should be easy to make, and the initial costs as low as possible. Having a single site for managing licenses, and thus the upcoming invoices, for all software provided by the vendor is a good selling point. The licensing system (probably a web site) should also provide access to downloadable parts, like client side applications, of the provided services.

Usage of the management interface needs to be very easy, but on the other hand it gives large control and responsibility. It should not be possible for the customer to make licensing or other administrative changes that would render the licensing framework unusable, such as removing the highest level administrative account or critical system licenses. Safeguards are discussed in Section 5.1.

The responsibility for monitoring license usage and promptly revoking licenses as needed lays on the customer. Unused licenses result in unnecessary costs, but software with expired or terminated licenses might not work at all. This means that the customer side administrators must have an understanding of what they are doing.

2.3.3 Customer associates

The actual licensing part is usually not relevant, or at least not particularly interesting, for this group of stakeholders. The most important thing is that they have access to the tools they need (or sometimes are required to use by the customer), and that "things just work". The system has to be easy to use and should always be accessible when it is needed. All licensing and authorization issues should be as transparent as possible, and preferably managed by someone else (in this case, eCraft's customer).

2.4 System requirements

The actual system requirements for the licensing framework can be specified after listing the stakeholders and their interests. This section presents the requirements, as identified by tasks and interfaces surrounding the framework.

2.4.1 Licensing

Software vendor goals

One of the main goals with the licensing framework is to ease the accounting work load. An integration between the framework and the existing ERP (currently Microsoft Dynamics NAV) should be implemented. Depending on license type, the number of active licenses or current pricing is readily available at the time of invoicing. Active licenses can be tracked on arbitrarily chosen intervals, which could depend on the customer and the software being used. A common value for the invoicing period is one calendar month, but pilot periods or other arrangements might need periods of weeks or even days.

It is sometimes necessary for eCraft to take care of license management, either on behalf of the customer or because of internal auditing, statistics gathering, security auditions and so forth. Among the interesting statistics data is license status and activity history. The latter could be useful for monitoring undesirable activities like intrusion attempts or even excessive license sharing in cases where sharing cannot be accurately prevented.

Some software makes use of configuration information, like database connection strings or role based restrictions, which is hidden from the customer and software users. These settings are managed by developers or possibly an account manager at eCraft and they are mostly project or customer specific. Managing license types, available roles and other configurations of the software must also be possible, and probably desirable, unless this information is hard-coded in the software.

For the framework itself to be usable by the customers, there has to be a possibility to manage customer users who can access the administrative interfaces. Since the administrative users have several privilege levels, it will probably be the responsibility of eCraft to maintain these levels and make sure they support the customer's organizational structure.

Software updates also need to be handled in the framework, and should be included in the normal license validity check logic for simplicity.

Customer goals

The customers need an interface to manage their licenses. The most important functions are cost prediction, license and role assignment to persons or groups, adding and removing licenses and reviewing usage history and other logged activities.

Cost prediction means that the license costs have to be displayed in real-time to the customer, who then can optimize the license amounts and types as customers are usually not willing to pay any more than they have to. Usage history and audit data are needed both to check that there are no unused licenses, and to verify that the software vendor (eCraft) is not invoicing any non-existent licenses.

Large customer companies usually have several departments or other logically or physically distinct entities, and thus might require invoicing information to be attached to a specific license. Internal budget conflicts are not uncommon in both smaller and larger organizations. This creates additional requirements for the CRM integration, and license management privileges.

Adding and removing licenses needs to be a quick and simple procedure. The customer has to get a feeling that nobody is forcing them to pay for unused licenses, and it is in the interest of eCraft to make sure it is easy to add new licenses immediately when needed by the customer, to maximize the revenue.

Managing license types and assigning them to users is also important, more on license types in Section 3.1. Assigning (software dependent) roles for users or license types is a frequently needed feature in many applications. This interface could easily be implemented in the framework, effectively removing one administration interface from the application itself.

There should be several levels of customer side administration access rights in the license management interface, which could correspond to the management hierarchy at the customer company or any other suitable hierarchy. Restrictions on license buying and role assignment would then apply in a way decided by eCraft. The customer might also want to set limitations on how many new licenses can be issued over a period of time, or otherwise limit the potential invoice growth.

Licensing goals

What happens if a license gets into the wrong hands? It is not unlikely that laptops get stolen or forgotten. In these cases, the user does not want to lose all access rights, but rather invalidate that specific license. The user or an administrator would then regenerate the license file, so that the old one is invalidated and thus cannot any longer be used.

The generation of a new license file for the user should be automated, and binding the license to a fresh installation of the software has to be easy;

drag-and-drop, double-click on the file or maybe have the file automatically included when downloading the software.

2.4.2 Software using the licensing framework

What kind of software and applications would use this licensing framework? Which features are needed? The foremost target and original inspiration for the framework is SaaS solutions, but any software that needs licensing, role– or configuration management should benefit from it too. Not all features needs to be used for every solution.

The goal is to have a framework that is completely independent of the software using it. This means that the interfaces cannot be tied to a specific technology like .NET. The current widely used standards like HTTP(S) and web services would probably be the best choice for portability and compatibility.

Customer side administrator

Most advanced software has at least some kind of built-in administration tasks. The tasks range from simply managing sets of more or less static values to handling more complex mathematical problems and scenarios. Common tasks are data administration, consolidation and reporting as well as aggregation of information to produce usable output for customers, vendors or retailers. Having different sets of roles for end users is fairly common, thus role and authorization management is important.

Some solutions, like in Scenario A (Subsection 2.1.1), handle lots of different data, which means that access rights needs to be defined in a dynamic way. This may include several levels of abilities, and maybe even having different groups on the same level. Some users could only handle user manual related tasks, while others handle the web shop data administration. Other solutions again, like in Scenario C, have a near flat administrative organization.

Customer subcontractor

Subcontractors need to get their design specifications and documentation sent to the customer, which means mostly data input and management of said data. Access should probably be somewhat restricted, unless the customer has full trust in the subcontractor. Defining application specific roles and tying these roles to the license should do the trick.

The software can either be a web application, an online or offline client program or a combination of these.

Customer retailer

Technical data and stock quantities are the most important retailer needs. Repair technicians often need specifications, drawings, repair guides, spare part lists and other data, including older revisions of said documents because sometimes equipment designs evolve during the product lifetime. There might not be any Internet connectivity when servicing (for instance, mining equipment deep under ground), making offline usage is a required feature.

Retailers and service technicians sometimes need to place orders for parts or products. Reviewing upcoming yearly service or other repairs is also important, although software dependent restrictions may apply (only certain data for the customers of the specific retailer should be available).

End user End users in this case are users who don't even need to know about licensing to begin with. The data they can access is either completely public, or possibly bound to a certain piece of equipment they have bought. In the latter case, a serial number or similar is needed to gain access to the data.

For scenario A, there might be a possibility to order service, check order status, maybe even make new orders. Warranty information and service history is of great value for expensive equipment, especially if the equipment is being sold.

Licenses may also be auto-generated on-the-fly, if the licensing scheme permits this. Scenario C is one example where this approach could be justified.

2.4.3 Software types

The licensing system must not be bound to any specific platform or environment. Some platforms imply additional requirements, like how licensing is checked and enforced for web applications. How about access from multiple IP addresses simultaneously? What is considered license sharing, and how to distinct it from a roaming or multihomed user? Roaming means that the connectivity changes during a session, like moving between different networks, and needs to be supported.

Most custom made software provided by eCraft is (at least for the moment) based on the .NET platform, and are usually written in C#. This is true both for client side, server side and web applications. Some are online only, some can be used both online and offline, and some are completely independent programs.

Several solution utilizing mobile platforms, such as Windows Mobile, are available, and will probably increase in numbers and popularity in the future. Mobile applications are by design usually offline capable, to make sure they continue to work when there are network problems.

Many existing real-world customer solutions make use of several technologies or platforms. There are intranet-like (for instance Microsoft Office SharePoint[1]) sites with Windows Forms workstation applications for data input, web shops with integration to an CRM or ERP, etc.

The framework needs to handle all aspects of both simple and complex solutions, which in practice means frontends, backends, plugins and middleware software. This suggests some kind of licensing module to handle the actual communication with the licensing backend, and a simple but powerful API that can be used by the software developers.

2.5 Scoping

There are lots of details and related tasks left outside the scope of this thesis. Invoicing related functionality is specified in a rather static way, partially or not paid invoices are left for further development as is grouping of licenses for use with separate invoicing addresses. The ERP interface is not designed for allowing new customers to purchase licenses due to practical limitations of the current system.

Exact API specifications are not provided, as the technologies used in the actual implementation probably would change them anyway. A functioning Certificate Authority is assumed to exist, but certificate generation and revocation is only briefly discussed. Only ClickOnce [22] is considered for software distribution, because it is the only practical update method currently used.

Debugger protection, virtualization detection and other protection mechanism implementations are not specified, as each one of them could easily turn into a thesis work themselves.

[1] `http://sharepoint.microsoft.com/`

Chapter 3

Software licenses

Software licensing consists of the actual license agreement and usually an enforcing component. The most important license types are described in this chapter, along with thoughts on license invoicing and management. Traditionally, a license has only dictated what the end user might use the software for, and imposed restrictions on how it is allowed to be used, once the user has bought the software. When selling a service however, the license also covers the time frame for use of the particular software. The software is actually being leased for a specific time, which usually is the same as the billing period. The lease is usually automatically extended, as long as it is paid for.

> *Software licensing is a contract of agreement between a software publisher and an end user of the licensed application.*
> ...
> *If you are a user you need to be sure what type of license you need and be careful not to abuse the basis of the license or you could be accused of "software piracy".*
>
> *Publishers who wish to enforce their software license tend to use other software applications to help them control their IP and these tend to be reffed (sic) to as "software license management" technologies.*
>
> *Gillespie-Brown, Jon. What is Software licensing?:A quick guide to the basics of Software Licensing by Nalpeiron [Internet]. Version 5. Knol. 2009 Mar 24. Available from:* `http://knol.google.com/k/jon-gillespie-brown/what-is-software-licensing/3v64x901bjfe2/2.`

3.1 License types

What kinds of license types are actually needed, in this kind of framework? The requirements vary drastically, depending on the type of software or service examined. Licenses can be personal, i.e. bound to a specific person (or possibly device, where the device in question is more important than the person using it), or assigned to some group of people, where a predefined number (or even infinite, if appropriate) of users can be active at any given point in time.

Group, or floating, licenses require a connection to the licensing server, at least for checking that there is a free license on startup and for reporting when the license is free again. The floating licenses can be bound to a specific user group, predefined login names or even let any user grab one.

3.1.1 Restrictions

Functionality or features in licensed software can be limited by the type of license, restricting which features can be used in the program. More fine grained limitations on what a user can and cannot do are then imposed by the roles the user has been assigned, controlling which features the specific user can access, and which data is available.

3.1.2 Usage count

If neither person nor device is too important, there could be a "usage count" type of license. Only usage statistics would be tracked, for informative purposes. This could be used when a license by itself has no cost, but the performed activities or number of users is of interest. The tracking data could be used to meter some application usage, to determine whether it's worth spending money on development of the application or if the usage is too low to support continued spendings.

3.1.3 Publication

One type of license and authorization could be "web publication" - data accessible using this license is available on a web site; either public or intranet. Sometimes, especially when integrating data to other systems, it might be

License type	Relevance
Usage count	Only used for tracking, automatically generated as needed
Publication or Data Connector	Could be used for web (inter− or intranet) published data as well as granting privileges required for system integration to the server.
Online−only	The "standard" license for online applications
Offline	Can be used offline for a certain period without update checks
Other	
Trial license	Not currently used, but should be supported for future needs
Custom	Where applicable, eCraft could insert custom license information

Table 3.1: License types and relevance

useful to define different access levels for publication licenses, which could be seen as roles.

3.1.4 Online-only

Most solutions offered are near real time interfaces to some data storage. This means that online only licenses should be considered the normal license. The enforcement of this type of client license is also the easiest, since any changes can be applied immediately. The client software is normally not usable at all (at least without having access to the customer network), and thus the client itself needs only light protection.

3.1.5 Offline use

Offline usage is one of the more interesting parts of this study because of the added complexity it introduces to the framework. Offline usage should probably be considered as a distinct license type. How can one implement strict enough restrictions and data security, while still keeping the software usable at the same time?

How long should the license be valid without checking the status from the server? Too long, and the license could have been terminated way before the software gets the notification. Too short, and the users will run into problems. For example, after a vacation, the user might not immediately be connected to the Internet (or intranet) for license verification, and thus the software might not work when needed. Any responsibility for loss of revenue or other inconveniences as a result of such circumstances should be specifically mentioned in the licensing agreement, as network connectivity might not even be physically possible in all locations.

3.1.6 Other licenses

A special case in licensing is a so called trial license. Most software using this licensing framework would probably not use such a feature at the moment, but there might be a future need for demonstration versions of software.

It would be possible to also include other types of licenses, than just for software published by eCraft. Hardware or other services could have their leases shown in the administration view, but software from other vendors (like Microsoft) should probably not be directly included since Microsoft already provides license management solutions for their own products, and it might not be a good idea (economically nor legally) to implement redundant services. These features are left for further development of the framework, as they are not directly SaaS offerings.

3.1.7 Active licenses

Invoicing is done based on the actual number of active licenses, but how should an active license be defined? The simplest definition is that it has been activated and has not expired nor been manually revoked. A login or application startup could count as activation for some time span (i.e., unlimited use for one week) or until it is released, as is the idea behind the floating licenses.

Invoicing could be done based on the number of user logins, application startups or certain actions performed using some software (kind of like help desk incidents, where a fixed price might be charged, regardless of how much effort is actually needed to solve the issue). Other scenarios could also be thought of, but this thesis limits the scope to manually activated and revoked licenses.

3.1.8 Licenses and roles

When binding licenses to software configurations (or roles), we introduce different levels and groups of authorizations, which could be seen as sublicenses or just more restricted licenses. Role bound pricing could be an option in some cases, where specific roles are significantly more (or less) valuable to the customer.

A license, personally assigned or floating, is always bound to a role, or a set of roles, in the software, unless the software is very simple and there is no need for separate roles. Enforcing roles is a straight-forward way of handling user restrictions, but it has to be implemented on both client and server side to be effective.

3.1.9 Module loading

Can the active license somehow be used for deciding which components of the software to download when installing or updating, or which to load at run time? All components might have to be installed anyway, if the same copy of the software is used by multiple users on the same device[1]. Can some components exist in code but be securely enough disabled for unauthorized users? This is easy on the hosted server side, but the client and any servers in the customer environment is more challenging.

3.2 License invoicing

An active license is required to use any of the software having license management, thus each license can be considered an income for eCraft, and naturally a cost for the customer. eCraft wants to make sure *all* use of the software is paid for, while the customer wants to make sure *only* actual usage is paid for. This suggests that transparency and accountability is crucial in the framework.

To receive the income for a license, an invoice is needed. How should licenses be tied to the paying customer? Is it possible or even feasible to handle invoicing on a per license basis? There might exist a need for dividing the invoice among several customer entities (departments, logical companies, countries etc), so that has to be easy to handle, both for the customer and for eCraft.

[1]Except with ClickOnce applications, where each distinct user profile will have their private copy anyway

3.2.1 Multiple customer entities

eCraft could send all invoices to one customer contact, who then distributes the payments internally to entities, resellers, subcontractors, etc. A subcontractor, though, is probably not too interested in paying for using some software that the customer requires them to use in addition to their "normal" software.

Invoices could also be sent to the actual license holder, where applicable. This adds a layer of complexity to the integration between the licensing framework and the ERP, and is probably not feasible from eCraft point-of-view. The customer on the other hand would have to care less about the costs involved, but still be able to follow up on usage and auditing data through the license management interface.

3.2.2 Expiration and revocation

Sometimes, invoices are not paid on time or at all, or use of the software could continue after the license has expired (if permitted by policy). Other unlicensed activity should also be detectable, as it is not unlikely that someone eventually will try to crack the licensing logic. What should happen in these cases?

Is it OK to somehow cripple the software? There is a great risk of having upset and angry customers, especially if the software is crippled by accident. In the worst case, could this result in lawsuits and damage claims?

Authentication and authorization is not worth much without having auditing of the user activities as well. Audit data can be used for many purposes, both required legitimate and possibly shady ones[2]. The importance of detecting possible fraudulent use depends on the customer and the data classification, but generally customers want some kind of reports of activity, and checking for unused or unnecessary licenses is probably going to be popular (who doesn't want to save on extra costs?).

3.2.3 Discounts

Customer specific pricing and discounts based on active licenses could be implemented when the amount of licenses grows. Ideas for rules can be found

[2]This thesis focuses on enabling the legal purposes

in Apple Volume Licensing Programmes[3] or Microsoft Volume Licensing [9] documentation.

How should special invoicing rules be defined (and where)? There could be one or several ranges of license amount discounts, each changing the billed sum in a certain way. The rules need to handle at least fixed costs for fixed ranges of license numbers and thresholds for price reductions, possibly also something like "buy 10, get one free". Combinations of several rules could also be needed in some cases, making the discount system non-trivial.

Fixed prices

0-5	$5x$
5-15	$12x$
16-30	$20x$

Arbitrary discount ranges

0-8	nx
9-26	$8x + \frac{34}{39}(n-8)x$
27+	$8x + \frac{34}{39}(18)x + (n-26)\frac{2}{3}$

Static -10% per license from last discount range

0-5	nx
6-20	$5x + (n-5)\frac{9}{10}x$
21-50	$5x + 15\frac{9}{10}x + (n-20)\frac{9}{10}\frac{9}{10}x$

Table 3.2: Sample discount rules. n being the number of licenses and x the base price.

[3] http://www.apple.com/uk/software/volumelicensing/

Chapter 4

Proposed architecture

The reason for protecting software from unwanted use is usually piracy. eCraft does not supply software that is of any real practical use on its own. Instead there is some kind of client software or interface to the customer data. This means that there is no use in trying to prevent all copying of the software, but rather make sure that the customer is invoiced for the usage, and that the customer data is secure; i.e. an arbitrary copy of the software should not be usable, unless the license is activated and permits use.

4.1 Overview

Having user and role administration separated from the software licensing administration interfaces is not very efficient. The tasks are very similar, making it a natural choice to combine them. User and role management is something natural in most companies, whereas license management might be seen as an unnecessary, unproductive and time consuming burden. Making the licensing almost transparent and automated is therefore positive, as there are almost no additional maintenance costs incurred in contrast to needing eCraft efforts for trivial installations.

The framework enables eCraft to manage tenants (see Subsection 4.7.1), available software, roles, configurations and billing. The roles are usually coded in the software, so the developers just need to add the roles, their descriptions and hierarchy to the licensing framework. This very simple role management should be sufficient for the first version of the framework, but could later on be extended to use something more advanced like XACML [17][24] or integrating more environments and adding trust management [21].

The customer chooses how many licenses, or users, he needs, and then allocates them accordingly. He also assigns roles to users (or licenses where applicable), monitors activities and invalidates licenses as needed.

For client applications, a license specific one-time, or at least time limited, download URL can be generated through the administrative interface, or the license file can be downloaded for manual forwarding to the correct recipient. Automatic mails could be sent out to the designated user as long as the address is known and security policies are not broken.

The end users should normally not have to care about anything; manually updating the license at worst. Problem cases can be handled by the customers own IT support or other administrative entities.

4.2 Non-practical ideas

Looking for an existing solution for handling licensing issues resulted in a list of quite a few methods. None of the solutions did however fulfill nearly all of the specified requirements, but they do provide ideas for a custom implementation.

4.2.1 Hardware locks

Having a hardware lock (commonly known as a "dongle") for software protection is common among some software types like CAD[1], DAW[2], printing and publishing. The software itself is usually quite expensive, which leads the manufacturers to fight piracy using any means available. The more advanced dongles can contain licensing terms or even parts of the actual software in addition to just an encryption key, for data access, that the simpler ones provide.

Hardware locks could be used if the software was really expensive and copies were limited to being used on "normal computers"[3]. The cost would be way too high and usage too cumbersome for this licensing purpose, and since all software in eCraft's case is distributed electronically (and should work almost immediately because customers rarely plan for installation well in advance), it's not feasible to use hardware locks. According to Aladdin Knowledge Systems [2], a hardware lock costs anything from €10 to over €100 per lock.

[1] CAD - Computer Aided Design. AutoCAD, Pro/ENGINEER etc

[2] DAW - Digital Audio Workstation. Software/hardware used at recording studios etc.

[3] Most dongles nowadays are USB sticks.

4.2.2 Conventional software locks

Several solutions exist, among others HASP [1] and CrypKey [5] that enable similar functionality to dongles, but in software. These solutions might be a bit cheaper [2] than the hardware based (with prices starting at a few thousand euros per software or SDK). There are options to either simply protect the software after it has been compiled, or use an SDK to implement the protection by embedding it in the source code.

All solutions come with some kind of cost, either a yearly fee, or an expensive SDK; CrypKey $4495 per standard SDK, Enterprise version starting at $4995 per installation. Only the Enterprise version has most of the features needed, but it's still not quite good enough. Several other solutions for very static licensing, aimed at consumer type applications also exist, but these solutions cannot quite adapt to the requirements for his study; the license is only bound to a specific computer, and do not support multiple roles per installation in the dynamic way needed. License management is usually also done only by the vendor as opposed to the customer approach needed here.

4.3 Conventional client-server architecture

One of the main goals of the licensing framework is to make it suitable also for already existing solutions, mainly conventional client-server architectures. This is achieved by adding the licensing module on both client and server side, and changing some of the code to use the provided features.

4.4 SaaS

One of the points of having the customers manage their own licenses is selling software as a service. Another equally important point is that by licensing software instead of just selling it for a one-time fee, the revenue can increase significantly for the vendor, while still keeping the customer happy [25]. This is achieved by continuous updates without explicit fees for upgrading and letting the customer decide how many licenses they need during any arbitrary time intervals.

The framework has to take into account that new customers someday might register directly, instead of having eCraft manage all customer details, but such a feature is not needed in the near future. The self-service approach

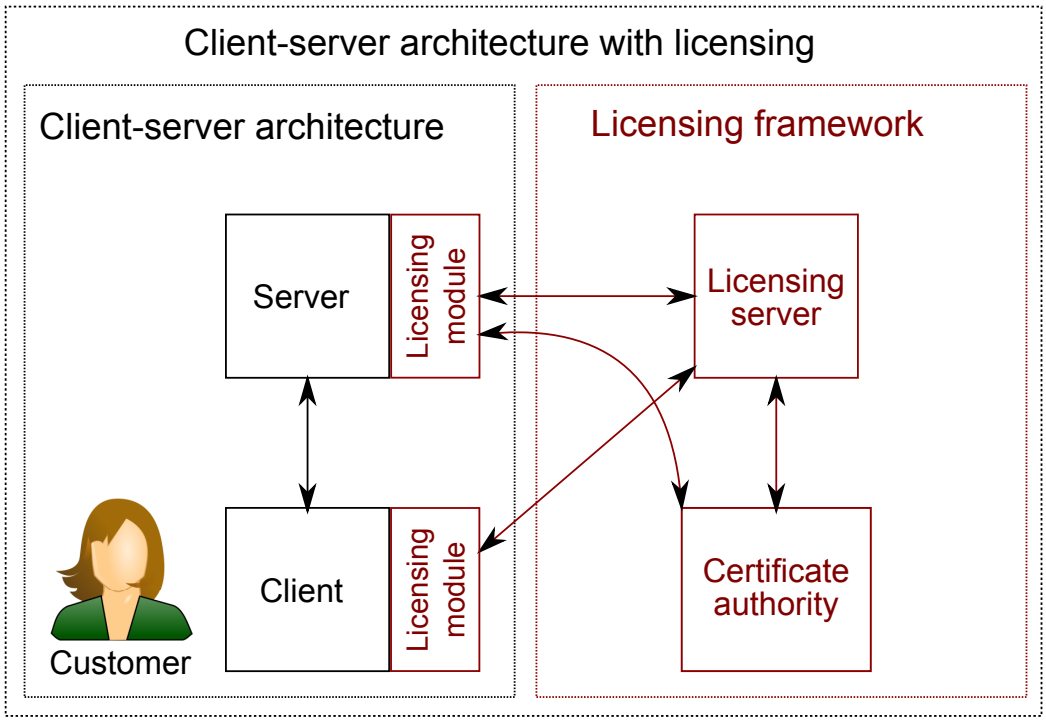

Figure 4.1: Conventional client-server architecture with licensing (depicted in maroon) added

could also be used to provide trial versions of some software.

4.5 eCraft administration

eCraft tasks can be divided into three main categories. The first category contains all tasks related to managing software in the framework, the second is tenant management and the third contains tenant-software-management tasks.

4.5.1 Software management tasks

Adding software to the framework requires some basic information like the software name and path to installation files (or URL of the web site).

Possible configurable settings, usually at least database connection strings and available roles (including role hierarchy), have to be defined.

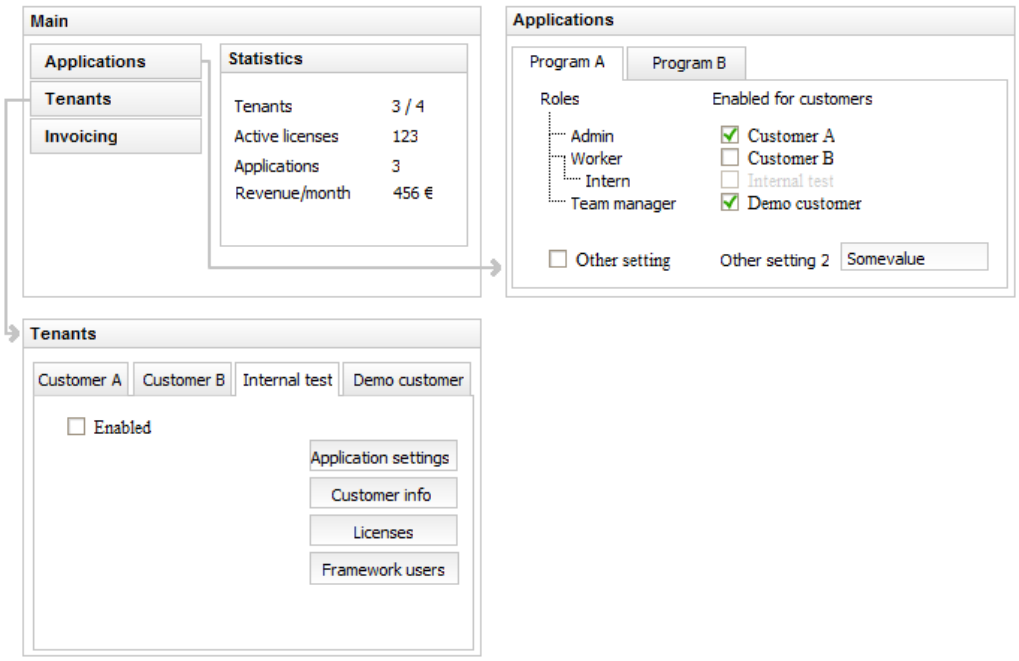

Figure 4.2: eCraft administration interface mock-up

The framework relies heavily on certificates, and thus there needs to be functions to interact with the CA to link a certificate to the software, generating a new one if needed.

4.5.2 Roles

There are several different ways to implement user roles in software, although the only solution supported by the framework to start with is the very simple one. The others could be implemented at a later time, if needed, but at this time, they are not used in any real projects.

Very basic role management

The software developer just lists a list of available roles (administrator, manager, user, ...), possibly in a tree structure where a selected role can implicitly apply all descendant role too. The application has a function like bool HasRole(string roleName) in the licensing module, which then checks the license file for existence of the specified role and returns the result.

This approach is not very secure, since it can quite easily be bypassed by altering the binary. It is however usually considered good enough, especially if the software in question has a server component, where the server also checks the user privileges.

Active Directory Application Mode

Microsoft provides a feature called Active Directory Application Mode (ADAM) as an extension to Active Directory that provides Lightweight Directory Access Protocol (LDAP) functions in user mode.

ASP.NET Roles

ASP.NET provides role management out-of-the-box[4]. There are several backends (Role Providers) shipped, and it's fairly easy to create a custom one[5]. This is limited to web applications though, which is also implied by the name.

4.5.3 Tenant management tasks

Adding, updating and removing tenants are the basic tasks. Tenant information needs to be kept up-to-date, either manually or by integration from the ERP. To start with, the tenant name and ERP customer number are needed to enable invoicing.

The different customer specific administration levels for the framework are also managed here. Level-specific restrictions may be defined.

Checking billing information, like old and upcoming invoices, is also important. The ERP integration automatically marks paid invoices, so it's easy to identify anomalies here.

4.5.4 Making software available for tenants

Most software will need customer specific configuration and licensing information. A link between tenant and software has to be established, and the configurable settings have to be defined.

[4] http://msdn.microsoft.com/en-us/library/8fw7xh74.aspx
[5] http://msdn.microsoft.com/en-us/library/tksy7hd7.aspx

Defining how the software is to be downloaded is also managed here. The URL can be either static, or dynamically generated if needed. The framework acts as a proxy for getting the files; ClickOnce updates work as expected, but the framework decides which files are actually served instead of just serving static files.

4.5.5 License management

The available license types will be more or less static, as defined in Section 3.1. All types of licenses are available to all programs in the database mock-up at this point, even though it probably would be better to define per program types.

Adding licenses which are read only for the customer could also be done in some cases, for example when a certain server component in the solution cannot be removed without effort from eCraft.

4.6 Customer-side administration interface

The customer-side administration tasks include managing user privileges for the licensing framework interface, in addition to adding, removing and assigning roles and licenses. It should be possible to make any wanted changes and view them somehow before actually committing them, though such a feature is up to the implementation of the interface.

4.6.1 License administrators

Since there can be several levels of administrators, they have to be managed. This is naturally something eCraft could do, but the customer top level administrators probably want to do it themselves, to save money or time. Framework logins can naturally be shared among the users, but having a personal login makes audit data a lot more useful.

4.6.2 License and role management tasks

Assigning users to license types and defining user roles are the most common tasks for client-side license administrators. When a user needs to reinstall the software for some reason or changes his computer, the administrator

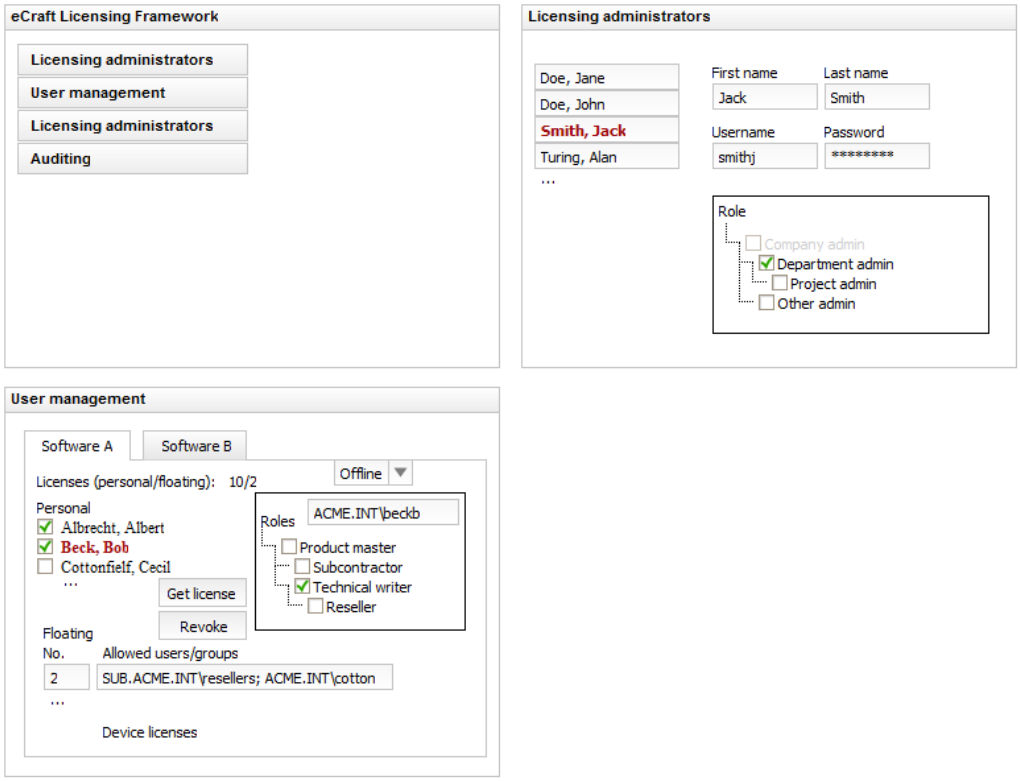

Figure 4.3: Customer administration interface mock-up

simply generates a new version of the license, automatically invalidating the previous one. This means that the license has to be updated again on the original computer, even if the change is temporary. Sometimes it might be easier to just generate a new temporary license.

4.6.3 Other tasks

It might be desirable to define limits on how many new licenses can be bought during some time period. The spending limit can be defined in currency or number of new licenses per week or month. Higher level administrators or eCraft could define these rules.

4.7 Database

The backend storage for the framework is a relational database, Microsoft SQL Server. The database stores all information the frameworks needs, except for the software files, which are stored in the file system. The database is designed with security considerations mentioned in Subsection 5.5.1 in mind.

The database contents can roughly be divided into three segments; common data, eCraft data and tenant data, which are described in Subsection 4.7.3 through 4.7.5.

4.7.1 Multitenancy

Multitenancy is a principle in software architecture, where several clients (tenants) can be served by a single server instance.

There are several reasons for using multitenancy instead of completely separate server instances. The most important ones are cost savings through reduced hardware requirements and easy data aggregation. Releasing updates is also simplified, as the database and code changes typically only needs to be distributed to a single location.

4.7.2 Tenant authentication

The design for how administrative users are handled presented here implies that the user privileged are checked directly against the database as shown in Figure 4.4. This way, the schema is automatically applied, without the need for a common administrative users table readable by the login service. The benefit of having a simplified authentication check has some potential problems though, most significantly the point that SQL logins have to be created in addition to listing the user in the administrative users table in the customer schema.

4.7.3 Common data

The common data consists of license types, base pricing, available software information, roles and configurable settings for the software and possibly other miscellaneous data like translation strings etc. Table details in Section A.2.

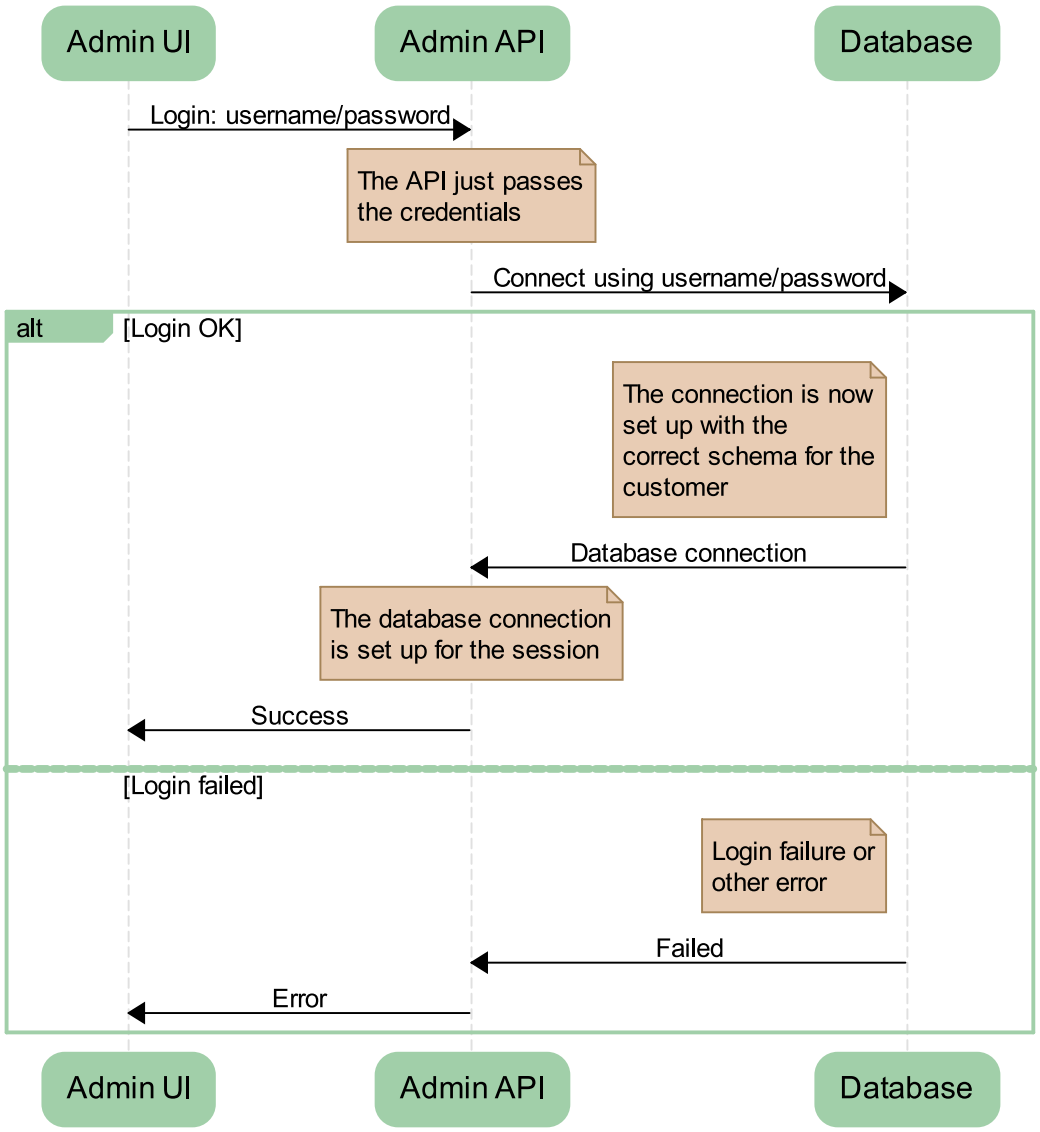

Figure 4.4: Customer side administrator login

4.7.4 eCraft schema only

There is not much data available only to eCraft in this design. The only part that customers never need to see or touch is related to who the tenants are, and their link to the invoicing system. Table details in Section A.3.

4.7.5 Tenant schema only

There is a lot of important data that is specific to a customer. License information, software information, current and historic usage data, version numbering and certificate details act as the core information.

The program specific settings as well as license bound role information are used for generating the actual license file.

License usage is aggregated as defined by invoicing periods into the invoice table, where the invoicing integration picks up information for the ERP. Invoice payment status is then updated back to this table as invoices are marked as paid. License information will not be deleted from the license table, so any history data can be found there as well.

Any actions done through the licensing framework administration interface is logged in FrameworkLog, and any API actions performed by either server side or client side applications are logged in LicenseLog table. All table details in Section A.4.

4.7.6 Pricing

Each program should have a generic price set, which is identified by a null tenant reference. Special pricing rules, with multiple rules per license type, can then be applied by eCraft for each tenant where needed. A price rule also defines the billing period and how many licenses are included in the base price. There is also a possibility to define a license price multiplier for each role. This means that the final price for a license is calculate as $max(Role : LicensePriceMultiplier) * License : Price$. The multiplier should be kept inside a reasonable range, maybe 0.1-2.

If changing a role also changes the price, a new revision of the affected license needs to be created in the license table. If the license is activated, this means that the cost for said license changes immediately, otherwise nothing happens until the license is activated.

4.7.7 Tracking and auditing

For both customers and eCraft, tracking as much as possible data about license changes, usage, configuration changes etc is an important part of the framework. Some tracking can be done on web server level, but most would be done in the database.

4.8 Core functionality

4.8.1 Certificate management and verification

The license file is signed partly by the client before activation, and fully by the server when activating the license. There has to be some mechanism to generate the keys where needed, and verification of signatures has to be easy. Using a Public Key Infrastructure works exactly as needed; the only problem with a PKI is how the certificates are exchanged. In this licensing framework design, the clients get their certificates embedded in the license file. Embedding the keys should not be a big security problem, since the license file is supposed to be kept secret anyway.

Certificate authority

Using PKI effectively requires the use of a Certificate Authority (CA). Setting up and configuring the CA is beyond the scope of this thesis, but there are several solution for this, ranging from cheap[6] to expensive and simple[7] to very advanced[8]. The only requirements are an API for the licensing framework server to manage client certificates, and a working Certificate Revocation List function.

4.8.2 Client application

Client application will access all licensing functions through a licensing module (described in Subsection 4.10.2). The module interacts with the backend through the client web service functions as well as provides internal functionality for the application.

Methods that contact the licensing server are LicenseUpdate and KeysDestroyed. Methods that interact with the license information include initialization checks, role checks, configuration loading and transparently connecting to local encrypted database files.

[6] Open source OpenSSL, OpenCA

[7] All recent Microsoft server versions contain CA software

[8] `http://www.microsoft.com/forefront/identitymanager/`

4.8.3 Server components

In addition to the client methods, a few server side only methods are needed in the cases where a server component exists in addition to the client component. The server needs to verify the client certificate and possibly the client license validity whenever a connection is established, since only the backend (and the CRL) are aware of revocations.

Connections between the client and the server can now easily be encrypted and authenticated using the certificates. Most client-server solutions do already encrypt the traffic in some way, at least the ones using web services, so the change should be trivial to implement.

The server interface is also to be used for web sites and similar solutions, where an explicit client component does not exist.

4.8.4 License distribution

Distribution of the license should be very simple and flexible, probably meaning a stand-alone file. A small file can be emailed, transferred on a USB stick, sent over bluetooth, downloaded either as part of the original installation, through an update or separately via the license administration interface etc.

Sometimes it might make more sense to just get the license download URL, which would be usable once only, instead of the actual file. Including the license in the actual download means some overhead on the server side, as (especially with ClickOnce) the download manifest needs to be recalculated when changing or adding files to the download. The easier way would be to have a generic download, without license information, and then get the license separately to activate the installed copy.

Most existing licensing systems seem to rely on some kind of activation for a license. This activation usually fingerprints the hardware in some way, and registers this fingerprint with the license server. This is only useful when the software in question can be locked to a device. The software could still be used by multiple persons simultaneously on the specific device.

4.8.5 License file structure

The license file is based on XML, since it's an easy way to get both machine– and human readable, structured content and it is well supported by almost all platforms. Normal users don't need to look at the contents, but it's nice

to have for troubleshooting when problems arise. The embedded signatures will naturally not be verifiable in a generic text editor, but the rest of the interesting data is. The client program will take care of the verification in due time. See Appendix B for sample file.

Sensitive data (if there is any, connection strings for local DB:s should use integrated security where possible anyway) is encrypted with the client public key.

The license activation is performed in two steps. After attaching the license file to the application, the client generates the hardware fingerprint and other client specific data, signs it and sends it to the server. The server then signs the whole license file, which now includes the fingerprint, and returns it to the client.

4.8.6 License verification and update

The licensing module takes care of updating the license information. Depending on how the license is actually stored (only in file system to begin with), the client program will need write permissions on the file. This is not a problem for client applications, but web applications should not have writable files.

Figure 4.5 describes the communication sequence when performing a license update. There are several possible outcomes of the update check: everything is OK, the license is revoked or there are no available licenses. A revoked license means that an administrative user has permanently disabled the license, and the client should erase its keys if possible. Using floating licenses might produce the last result, when all licenses are currently in use.

At least for offline enabled clients, the license will be updated as often as possible, within reasonable limits, to maximize the validity time frame if the network connection gets interrupted. Other clients will also check the validity at regular intervals, but the actual license does not need to be updated until the validity time has expired (as the application is online-only anyway).

4.9 Billing integrations

There is no need to explicitly keep track of billing status in the framework from eCraft point-of-view, since billing is performed elsewhere anyway. However, it might be of interest to the customers to be able to see this information

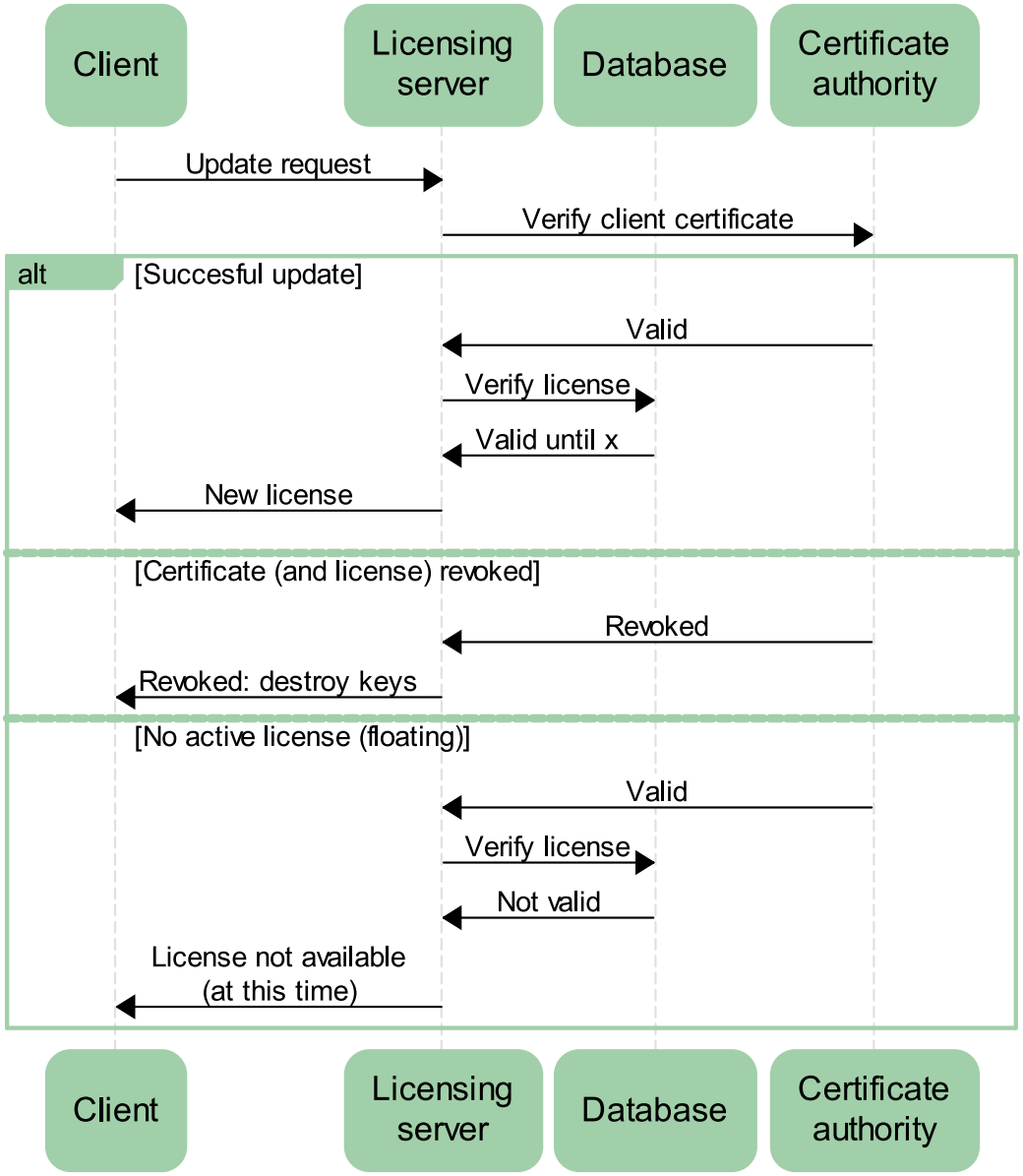

Figure 4.5: License update sequence diagram

through the administration interface, and therefore it is included.

The software types taken into consideration here are all of such type, that there will already be an existing relationship between eCraft and the customer. This means that new customers won't have the possibility to sign up for SaaS offerings at this time, but it will probably be implemented in the

future.

4.9.1 Database requirements

By looking at the customer billing period defined in the framework, selecting all licenses which were active and combining these with the license prices, we get the sum to invoice. In the scope of this thesis, this is how far invoicing related matters are handled. I will therefore not consider partially paid invoices, since the business requirements for such cases might change anytime and they might also be very customer, software or project specific.

Multiple invoicing addresses for a customer is not possible at this time, due to the fact that a customer in the licensing framework equals a single customer company in the ERP. This means that a separate customer, and thus also separate administrators, has to be set up in the licensing framework for each required separate billing address. Changing this behavior would require modification of the ERP system, which is not desired at this time.

4.9.2 Integrations

The integrations are to be implemented in BizTalk [11], since it is already established as the primary platform used by eCraft for business integrations.

```
1 foreach tenant
2   if last Billing_period < now − Billing_period
3     create new Invoice
4     link Licenses to Invoice
5     fetch Invoice data into ERP
6     get ERP Invoice_number
7     update Invoice with Invoice_number
8   endif
9 done
```

Listing 4.1: Pseudo code for invoice data integration

Licensing framework to Navision

The integration logic, as seen in Listing 4.1, is quite straight-forward. Future improvements could include advanced license grouping and license specific extra information.

Navision to licensing framework

The licensing framework needs to have some basic customer data synchronized with the ERP. The information is very limited at this point and only includes customer name, ERP customer id and information on whether or not the customer has access to the licensing system at all. This synchronization is easily triggered on customer information updates in the ERP, as daily batch transfers or the data could even be entered manually to begin with.

Another integration, Listing 4.2, fetches invoice payment status for update to the licensing framework.

```
1 foreach tenant
2   if exists new paid invoices
3     foreach Invoice_number where status = paid
4       update Invoice set status = paid
5     done
6   endif
7 done
```

Listing 4.2: Pseudo code for invoice payment status integration

4.10 Licensed software

This section describes the client software parts of the architecture. The easiest way to add licensing capabilities to some software is probably by adding a licensing module, and then have the module initialize the actual application. The module also takes care of handling the license. Database connections (mostly to local embedded databases) are also established through the module, for transparent encryption and decryption.

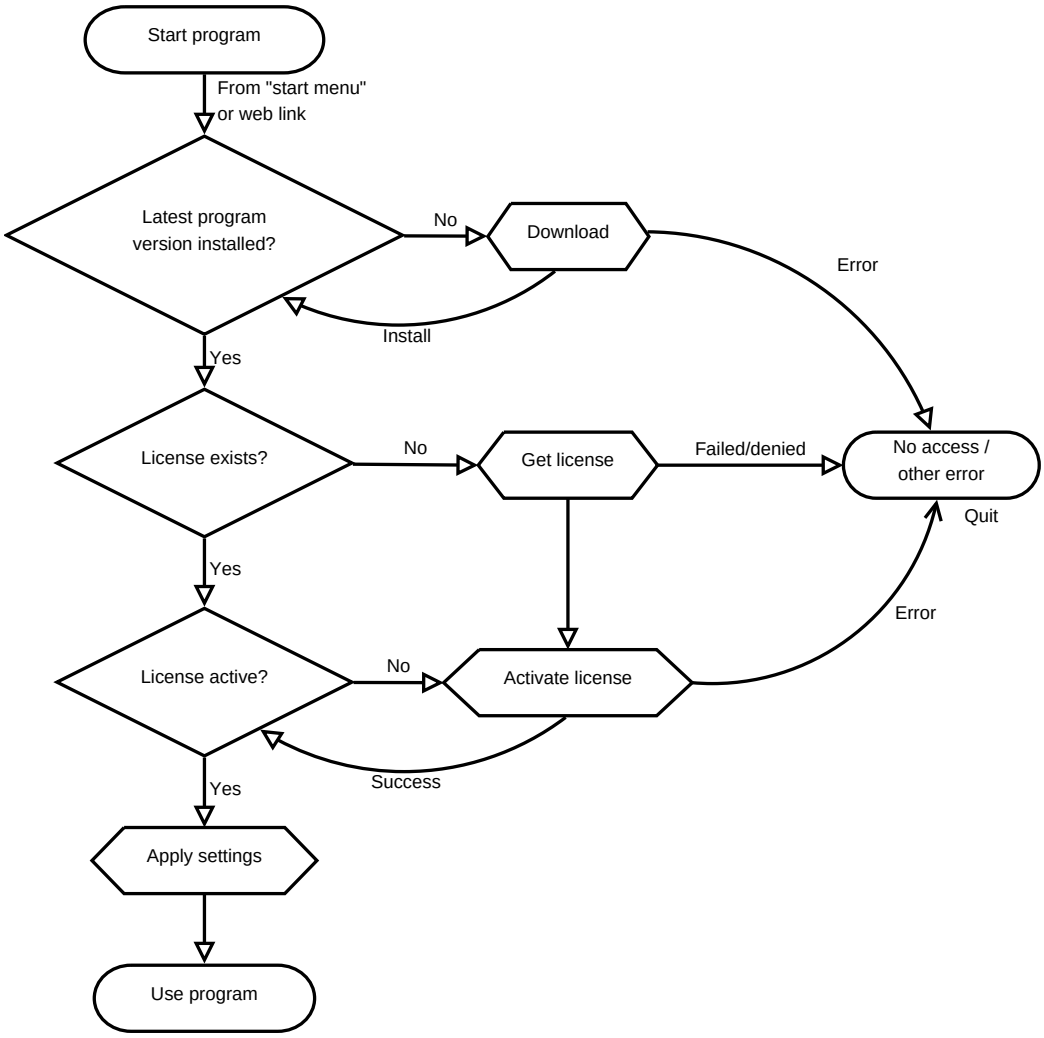

Figure 4.6: Client software start/install using ClickOnce and licensing framework

4.10.1 Application types

There are several different types of applications, and most of them should benefit from the licensing framework. The types described here have strong ties to the Microsoft .NET world [6], but similar types exist for other frameworks and technologies also.

Windows Forms

The standard client application type is Windows Forms application, mostly Smart Clients with ClickOnce deployment [22]. This category also includes Silverlight[9] applications, since they are no longer web only. These clients can be either thick, thin or smart.

Thick clients have all functionality deployed locally and are offline capable or might not even include any connectivity functions. All licensing related functions are client side only and the license can be valid for long periods of offline use.

Thin clients are online only applications, that usually have no offline capability and needs a browser or similar to function. License on server side only.

Smart clients again, are a combination of the other two, meaning there can be a rich user experience through local resources, while still having the benefits of an online application; they are both easy to deploy and update.

These applications can be either stand-alone, client-server or even use peer-to-peer communication. Licensing requires the licensing module to be part of the client and, where applicable, the server.

Web applications

Pure web applications reside on the server only, so the licensing has to be done there. This means that licensing is easy to enforce, but requires the server side code to fully implement the licensing.

Web services

Web services are used in client-server or server-server communication. The same reasoning as in the web application server side logic applies. Both the customer and eCraft administration interfaces are designed to be used through the web service API, so the frontend is independent of the actual technology behind the service. The first version is probably a web application,

[9] See `http://www.microsoft.com/silverlight/`

but might be integrated into some other platform (like a CRM) through VSTA later.

Licensed applications and licensing proxies will also use web service calls to communicate with the framework server.

Smart device applications

There are already several solutions where a mobile device is the platform[10] on which the client runs. The .NET Compact Framework is used here, so the code for the licensing module should be .NET CF compatible[11].

Windows services

When there is a need for background operations that are not feasible to implement through scheduling of jobs[12], a Windows service[13] [14] is a good choice. Licensing works mostly as for a client application.

Hosted applications

A hosted application is like a normal application, except that it is contained inside another application. An example of this is a plugin for Microsoft Office Outlook. The applications are created through the Visual Studio extensions "Visual Studio Tools for Office" (VSTO) and "Visual Studio Tools for Applications"[14] [10] (VSTA), but from the licensing point-of-view, they are client applications.

4.10.2 Licensing module

The module is the central part of the client side licensing implementation. It provides methods for interacting with the licensing and CA server and other needed protection functions.

[10] Windows Mobile `http://www.microsoft.com/windowsmobile/en-ca/meet/version-compare.mspx`

[11] The CF includes only a subset of all .NET features

[12] Scheduling can be done as tasks in windows, cron jobs on *nix, SQL Server Agent jobs etc...

[13] Daemon in *nix

[14] VSTA can be used for add-ins to COM and .NET applications

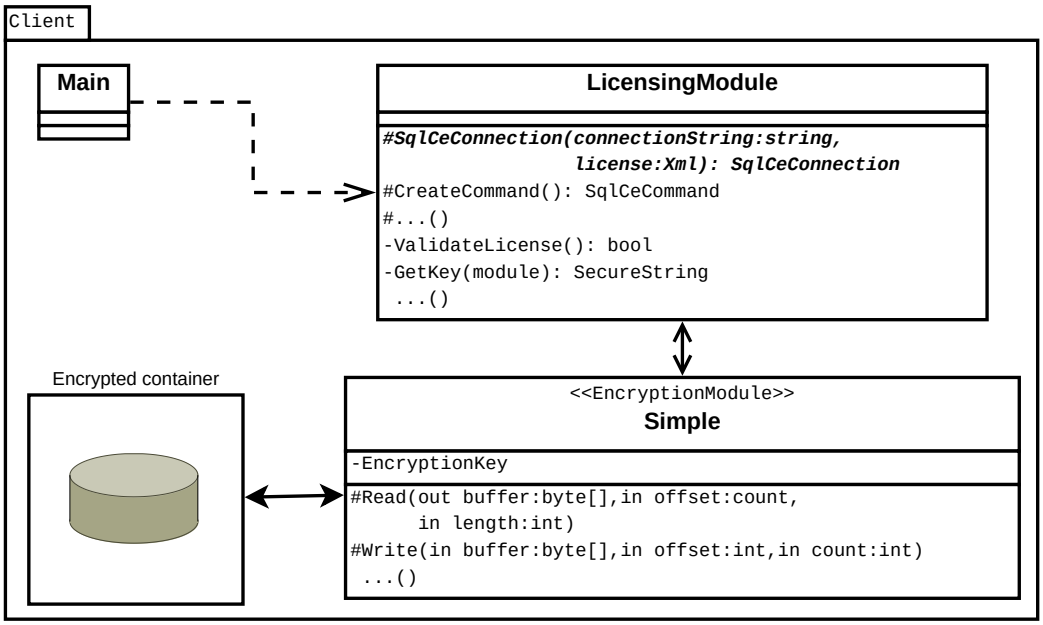

Figure 4.7: License verification and DB encryption module

Some advanced functions that the module should perform are beyond the scope of this thesis and part of future development. Such functions are, among others, detection of virtualized environment, real secure key storage, tamper resistance and clock change detection.

Functions

The module is required to start the software, and it checks the license validity each time as described in Figure 4.6. If no license is found, there should be an error message logged for server components, or a dialog shown for client applications. The dialog could take the license as a dropped file or have a text area for pasting the file.

Activating the license includes getting the hardware fingerprint of the device, the current user name and possibly other identifying information (see Subsection 5.3.1). This data is then included in the license file, signed with the client key and then sent to the licensing server for activation. Lastly, the full, activated, license file is received and stored.

Client configuration files can have their sensitive data encrypted with the client public key; the server knows the public key, and each configuration

can be marked for encryption. The client should generate a suitable key for encrypting local data. Some well known block cipher would probably be the most suitable for transparently securing random disk accesses.

4.11 Licensing proxy server

There are situations where a direct connection to the eCraft hosted licensing server is not possible from the customer network. This means that a proxy server should be installed on a server managed by the customer adding some complexity to the framework; the data handled by the proxy and the server binaries needs to be protected against tampering. Some data needs to be cached to provide the most important functions even if the proxy temporarily loses connectivity to the main backend, but the CA functions cannot securely be provided by the proxy.

To the applications, the proxy server looks just like the real backend and the proxy URL can be specified in the license file.

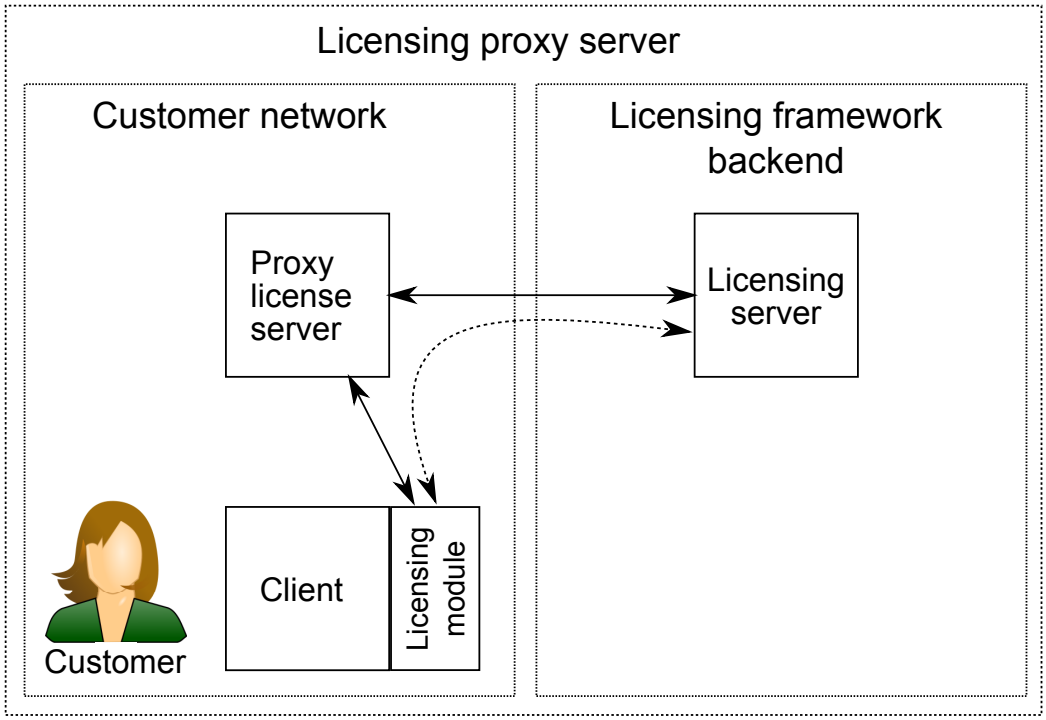

Figure 4.8: Licensing with customer side proxy server

Chapter 5

Security

Security is not something that can easily be applied once something is finished, but in this chapter I try to point out some of the most important security aspects of this licensing framework solution and how they are handled. Perceived security is not always the same as actual security, but rather a result of risk assessment, which is discussed in Section 5.8.

Failing security can be devastating, both economically and legally [18], and must therefore be thoroughly thought of.

5.1 Framework safeguards

The framework and associated interfaces need to implement some safeguards against both user mistakes and deliberate misuse. There will most likely arise other situations than the ones described here, but it is impossible to foresee all problems and new safeguards can be implemented when needed.

5.1.1 User mistakes

While managing administrative users in the framework, it should not be possible for a user to delete his own account. Also changing the administration level for oneself could be disabled.

Buying new licenses has a limit on either the number of new licenses, the maximum cost increase or both during some predefined time interval. Even if an administrator accidentally creates licenses of the wrong type and then changes them to the correct ones, no extra cost is inferred because they have

not been activated. Temporary role changes could be ignored if the new role data was never updated to the client, if the invoicing integration checks audit data too.

There might be situations where a customer side administrator decides to revoke or buy huge amounts of licenses at once, but it should be considered an unusual situation and at least a warning about this for both eCraft and the customer should be generated.

5.2 Public key infrastructure

Having a working implementation of a public key infrastructure is essential for building trust relationships between servers and clients. The alternative of just using shared secrets is more cumbersome to manage, and would compromise the integrity of the whole system if one client gets into the wrong hands, unless there would be lots of different encryption keys which again creates unnecessary overhead.

5.2.1 Encryption key strength

Any keys used need to be strong enough; the certificates should use 4k keys where possible, although some mobile devices might need smaller keys if there is very limited computing power or memory available. Security requirements for encryption change all the time, requiring stronger encryption to provide the same level of security in the future; the 4k key size and RSA encryption is adequate for now.

Usually when dealing with PKI and other web of trust solutions, there is the problem of whom to trust and to what degree. Since this framework is designed for use by one ISV only, who will also manage the Certificate Authority and thus also the certificates, there is no problem.

5.2.2 Certificate revocation

Once a license (and its certificate) has been revoked, there cannot be any communication between the licensed client and the servers involved. This is made possible by a Certificate Revocation List, which is managed by the CA and updated frequently. Clients that have an offline license could have a short validity time for the certificate also; the certificate can be renewed when

the client performs the license update check. It takes some time to generate new certificates, but the backend can probably have some new certificates pooled for every available program to reduce the update response time.

5.3 Client device

Simple protection mechanisms like preventing several instances of the same program are readily available in most operating systems, but more complex features like debugger and virtualization detection and protecting the client binaries against reverse-engineering require more work.

The client should also be as tamper resistant as possible, which can be implemented in several ways including cryptographic methods [8] and even methods used for copy protection, DRM and cheat detection in online games [16].

If there are several users of one device, it is not possible to verify the actual user at any given time. This could be a problem if role based security is implemented in the software and the users do not log out when finished, but it is not a problem with the licensing framework per se.

5.3.1 Device fingerprint

Identifying a specific device is important when dealing with device locked licenses. The licensing module needs to implement a function for gathering this fingerprint, which is then sent to the licensing server upon license activation.

The fingerprint depends on the type of device being used, but typical data includes serial numbers (BIOS, hard disk, processor etc), MAC addresses, OS version, service tags (usually found on laptops and servers). The system needs to be flexible enough, so that small changes (like a hard disk replacement using cloned drive) do not prevent the software from working. Naturally, all changes are uploaded to the licensing server.

5.3.2 Data confidentiality

There are several problems regarding data stored on the client, of which confidentiality probably is the most important one. Since the data should be usable if and only if the license is valid and the user privileges are sufficient, it

needs to be protected somehow. The natural way would be to use encryption, but how would this be implemented to actually meet the requirements?

The licensing framework module should help the developers with data protection, so that there is no need to re-invent the wheel for every new application. All data stored or cached by the clients should be accessed through the licensing module which provides transparent encryption. This provides some data confidentiality, but other features should also be implemented in the future. This kind of protection requires more advanced techniques than plain .NET cryptography [7][13], since any keys used must be stored securely. The user (or even administrators) should not have any kind of access to it.

Having offline data available means either synchronizing a (whole or subset of) database between the client and the server, or just downloading some parts of the data on demand and then uploading new or changed data to the server at every synchronization. The former can be achieved in .NET utilizing the Microsoft Sync Framework [12], and the latter is usually coded explicitly for each application.

It is possible to create a custom data provider for the .NET framework [4], which internally would handle encryption and decryption.

5.3.3 Data destruction

Stored sensitive data should be made inaccessible or even permanently erased once the license is revoked. The exact extent of the revocation measures has to be decided based on both usability end legal aspects [19]. It is near impossible to know how many copies of the client files or even complete disk images there are, but it might still be nice to have the client report successful key destruction to the framework backend.

Key storage on dedicated hardware makes most key management and security issues easier to implement, but since that is not a real option for this framework at this point, some other measures could be implemented. If the key is never stored permanently on the client, it cannot easily be copied, but this approach prevents offline use.

5.4 Servers at the customer site

In almost all client-server architecture solutions (and also when using a proxy licensing server), the server is located in the customer network, and thus

under complete customer control. Securing data that should not be accessible by the customer can in some cases be done by encrypting the data through a key provided by an external party [20], for example the framework backend. All client-server communication is secured by the certificates attached to the licenses, and the server component verifies the validity of the client certificate through the CA.

5.5 Licensing server

5.5.1 Multitenancy

Adopting SaaS or cloud services requires the customer to have a little more than moderate trust in the service provider. After all, lots of important and usually confidential data is to be stored "somewhere" in cyberspace. The consequences of this data getting into the wrong hands could easily be huge economic loss or worse for the customer, and it would be devastating for the service provider. It is not enough for the service provider to have confidence in the system, since it is the customer who has to be convinced. Therefore a solid architecture is very important, so that security can be proven. Naturally, the solution still has to be efficient and cost effective.

5.5.2 Database isolation

There is basically three different approaches to database architecture design [15], all of which theoretically would work well, but each also has some drawback.

Separate databases

The most secure approach includes creating separate databases for each tenant, effectively limiting each tenant to only their own data. There is however significant overhead implied on the database server[1] and the maintainability also suffers. Furthermore, having a centralized (eCraft side) management interface for each application becomes cumbersome. This approach could be used in rare cases, where there is absolute need for isolation, for example banking or government solutions.

[1] Where "server" is mentioned in this text, it usually refers to both single servers and clusters.

Access to any shared tables means cross-database calls, which is usually more resource intensive, but not at all impossible.

Shared schema

The most efficient way from a database server point-of-view is using the same database and schema (i.e., the same tables) for all tenants, identifying the tenants by an Id-column in the appropriate tables. This approach requires the least resources on the server, and is quite simple to implement. Making administrative interfaces (for eCraft) is easy, but it's also easy to create problems.

There are drawbacks with single-schema solutions though; if there is a need to customize part of a database for a specific customer or solution, the changes will apply to all tenants. Isolating the tenants from each other also requires significant effort from the developers. These are the main reasons why this approach is unfit for many multi-tenant scenarios.

Separate schemas

A more suitable method for keeping data secure is using the same database for all tenants, but with separate schemas. This reduces the overhead needed for backups and allows a single server or cluster to serve more tenants.

For shared tables, the lookups are being done inside the same database, but across different schemas. The only thing needed for eCraft administrative interfaces, is a list of the schemas used by the tenants, which should be fairly easy to maintain automatically.

From both performance and security point-of-views, this is probably the best approach for this kind of solutions. Everything should scale well, and still have most of the security benefits from using separate databases and most of the maintainability of a single schema.

Overall database security

Accessing the database should always be done with the least privileges needed, which means impersonation of the tenant, to use the correct schema, from the web server, which means a combination of impersonation and trusted subsystem approach.

5.5.3 Software updates

ClickOnce [6] is an easy way to provide software updates for client applications. Unfortunately, this presents a problem if diversity [3] is applied to the downloaded files. ClickOnce deployments are signed both each file individually, and the package as a whole [6, ch.7]. Also, storing different versions of the same file consumes disk space, so the solution here could be to both compile and subsequently sign the files on demand.

Compiling and signing on the fly is very resource intensive and also introduces significant delay in the update procedure. This method might not be usable in practice.

5.6 Protocol

The communication between clients or servers and the licensing framework is built upon web services (using SSL) and secured by the certificates issued to them. This makes the communication as secure as it can be using common current technologies. Therefore it is more likely that the data content could be compromised before the client sends it. This means that the actual data content should be inspected more closely; in practice the signatures in the license file have to be checked.

PKI is used to authenticate clients, which effectively means that a terminated license can no longer be used for connecting to the servers at all.

5.7 Attacks

This section enumerates some of the possible attacks on the framework and its components. Eavesdropping on communications without any prior knowledge of the involved keys is not feasible, though protecting the client side certificates is important. The license keys can be transported unencrypted, which cannot easily be prevented, and might thus be an easy target for an attacker.

5.7.1 Software distribution

Software updates using ClickOnce cannot easily be faked, since the files are signed with a code signing certificate by the publishing developer. Other

update methods might me more prone to attacks, but they are left outside the scope of this thesis.

Using other distribution methods than ClickOnce makes the distributed files open to attacks. The user cannot be sure that the files are unmodified. This problem cannot easily be solved and the licensing framework provides no features for securing files explicitly. If the software is distributed as an installation package though, the installer can be signed with the usual code signing certificate, and thus the software provider name shows up at installation time on newer Windows operating systems.

5.7.2 Local attacks

The framework does not provide direct protection against local attacks such as debugging and reverse-engineering, but the whole software package needs this kind of protection as described in Section 5.3.

Local data cache or stores are encrypted through the licensing module, but since the key has to be stored somehow on the client, it is up to the technical solutions provided by the module to make key extraction difficult.

5.7.3 Network attacks

All usual network attacks, including (Distributed) Denial of Service and brute force password attacks should be expected. Most advanced firewalls have some kind of features to mitigate such attacks, but auditing and other monitoring should detect the rest.

5.7.4 License file security

As stated earlier, the license file might be sent in clear text over insecure networks. The actual effects of acquiring a license file by itself range from limited information leakage to possible future attacks on communication between the intended client and the servers involved. If also the software files are acquired by an attacker, some information about the customer infrastructure might leak, and unless the servers involved require additional authentication, it gives the attacker access to the customer data.

5.8 Risk assessment

A risk assessment from both eCraft and customer perspectives needs to be performed, since the actual and perceived risks might differ depending on the party.

5.8.1 eCraft

The economical risks include loss of revenue because of bugs or logical loopholes in the framework or its components. Complete or partial failure of the framework would result in customers being unable to use the applications they are paying for, probably severe customer dissatisfaction and possibly liability for payment of damages.

Security flaws in the system leading to the backend being compromised would lead to leakage of some information about the customers. At worst, an attacker could upload malicious code to be downloaded as updates to the software provided. Monitoring of the system activities is crucial in detecting and preventing these problems.

5.8.2 Customer

The customers trust eCraft to provide secure and functioning software and thus keep their data safe. The risks of data getting into the wrong hands always exist, but the framework minimizes these risks by adding new security features even to existing applications.

The biggest risk is that the licensing framework fails for some reason, making the applications and services unusable. Ending up paying too much license fees is also a real risk, but the safeguards implemented should minimize the impact of unwanted licensing changes. Most other risks regarding software solutions already exist anyway, and are thus left outside the scope of this thesis.

Chapter 6

Conclusions

This thesis presented a proposal for a license, configuration and user role management framework based on the current and near future needs of eCraft, a small independent software company. The background was a need for license management for SaaS solutions, and the requirements were gathered as a part of my daily work, while planning new features and programs. The goal was to have the customer do the actual license management, both to ease the workload on eCraft side, but also to give the customer flexibility in acquiring and terminating licenses.

The different parties that might be involved in this kind of licensing schemes were first identified. After this, the different roles of said parties were studied, to get an overview of their needs.

Having gathered the requirements, I set out to find current best practices for creating the framework. The proposed platforms and technologies are mostly based on solutions for Windows, since eCraft is mainly focusing on Microsoft technologies, but the APIs are usable on most platforms.

Storing license and tracking data is best done using a database. The backend is quite lightweight, and only exposes functions through web services for maximal compatibility. Customer and eCraft interfaces can then be built upon any platform, but my suggestion is using web applications, as it makes administration possible from almost any physical location.

The security of this kind of framework has to be solid enough to satisfy both eCraft and customer needs. Possible security issues were discussed and the solutions and implications of these issues were presented. Using PKI for securing communications and signing licenses is still the best solution from a flexibility and security point-of-view.

Finally, there was some work done to make sure the framework can be easily integrated with the existing CRM system. The amount of data to be synchronized turned out to be fairly small, which should make the actual implementation quite trivial.

The next step would be to create a working implementation of the framework. For the client software parts, this also includes further studies on secure local storage and anti-tampering techniques, which was left outside the scope of this thesis.

Bibliography

[1] Aladdin Knowledge Systems Ltd. Sentinel HASP protection keys. http://www.aladdin.com/hasp/protection-keys-benefits-models.aspx. Retrieved Feb 19, 2010.

[2] Aladdin Knowledge Systems Ltd. Software security - latest trends in software licensing, September 2005.

[3] Anckaert, B., De Sutter, B., and De Bosschere, K. Software piracy prevention through diversity. In *Proceedings of the 4th ACM workshop on Digital rights management* (New York, NY, USA, 2004), DRM '04, ACM, pp. 63–71.

[4] Beauchemin, B. ADO.NET: Building a custom data provider for use with the .NET Data Access Framework. *MSDN Magazine December 2001*. http://msdn.microsoft.com/en-us/magazine/cc301611.aspx. Retrieved Jun 2, 2010.

[5] CrypKey (Canada) Inc. CrypKey SDK. http://www.crypkey.com/products/sdk.php. Retrieved Feb 19, 2010.

[6] Hashimi, S. Y., and Hashimi, S. I. *Deploying .NET Applications: Learning MSBuild and ClickOnce*. Apress, May 2006.

[7] Meier, J., Mackman, A., Dunner, M., and Vasireddy, S. Building secure ASP.NET applications: Authentication, authorization, and secure communication, 11 2002. http://msdn.microsoft.com/en-us/library/aa302402.aspx. Retrieved Mar 18, 2010.

[8] Michiels, W., and Gorissen, P. Mechanism for software tamper resistance: An application of white-box cryptography. In *DRM '07: Proceedings of the 2007 ACM workshop on Digital Rights Management* (New York, NY, USA, 2007), ACM, pp. 82–89.

[9] Microsoft Corporation. *Microsoft Volume Licensing Reference Guide*, October 2009 ed. http://www.microsoft.com/downloads/details.aspx?FamilyID=cc499fd9-4830-4d57-93a4-6ed263bad02e&displaylang=en.

[10] Microsoft Corporation. Visual Studio Tools for Applications. http://msdn.microsoft.com/en-us/vsx2008/products/bb933739.aspx. Retrieved Mar 14, 2010.

[11] Microsoft Corporation. Microsoft BizTalk Server product information, 2009. http://www.microsoft.com/biztalk/en/us/product-information.aspx. Retrieved Mar 2, 2010.

[12] Microsoft Corporation. *Microsoft Sync Framework*, 2009. http://msdn.microsoft.com/en-us/sync/default.aspx. Retrieved Mar 8, 2010.

[13] Microsoft Corporation. *Cryptography Reference*. http://msdn.microsoft.com/en-us/library/aa380256%28VS.85%29.aspx. Retrieved 12 Mar, 2010.

[14] Microsoft Corporation. *Introduction to Windows Service Applications*, 2010. http://msdn.microsoft.com/en-us/library/d56de412.aspx. Retrieved Mar 20, 2010.

[15] Microsoft Corporation, Chong, F., Carraro, G., and Wolter, R. Multi-tenant data architecture. *Building Distributed Applications* (June 2006). http://msdn.microsoft.com/en-us/library/aa479086.aspx. Retrieved Jun 2, 2010.

[16] Mönch, C., Grimen, G., and Midtstraum, R. Protecting online games against cheating. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (New York, NY, USA, 2006), ACM, p. 20.

[17] OASIS eXtensible Access Control Markup Language (XACML) TC. OASIS eXtensible Access Control Markup Language (XACML), May 2009. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml. Retrieved Jun 2, 2010.

[18] Råman, J. Contracting over the quality aspect of security in software product markets. In *QoP '06: Proceedings of the 2nd ACM workshop on Quality of protection* (New York, NY, USA, 2006), ACM, pp. 19–26.

[19] Samuelson, P. Embedding technical self-help in licensed software. *Commun. ACM 40* (October 1997), pp. 13–17.

[20] Schattkowsky, T., Forster, A., and Loeser, C. Secure storage for physically exposed web- and application servers. In *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)* (April 2006).

[21] Shafiq, B., Bertino, E., and Ghafoor, A. Access control management in a distributed environment supporting dynamic collaboration. In *Proceedings of the 2005 workshop on Digital identity management* (New York, NY, USA, 2005), ACM, pp. 104–112.

[22] Szpuszta, M. Designing Smart Clients Based on CAB and SCSF: Architectural Guidance for Composite Smart Clients, 2006. `http://www.microsoft.com/downloads/details.aspx?FamilyId=5F9A8435-1651-4BE2-956D-0446A89A7358&displaylang=en`. Retrieved 24 Jan, 2010.

[23] Vacca, J. R., Ed. *Public Key Infrastructure: Building Trusted Applications and Web Services*. Auerbach Publications, 2004.

[24] Xu, M., and Wijesekera, D. A role-based XACML administration and delegation profile and its enforcement architecture. In *Proceedings of the 2009 ACM workshop on Secure web services* (New York, NY, USA, 2009), ACM, pp. 53–60.

[25] Zhang, J., and Seidmann, A. The optimal software licensing policy under quality uncertainty. In *Proceedings of the 5th international conference on Electronic commerce* (New York, NY, USA, 2003), ACM, pp. 276–286.

[26] Zhao, Q., Zhou, Y., and Perry, M. Policy-driven licensing model for component software. In *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on* (June 2003), pp. 219–228.

Appendix A

Database diagrams

A.1 Overview

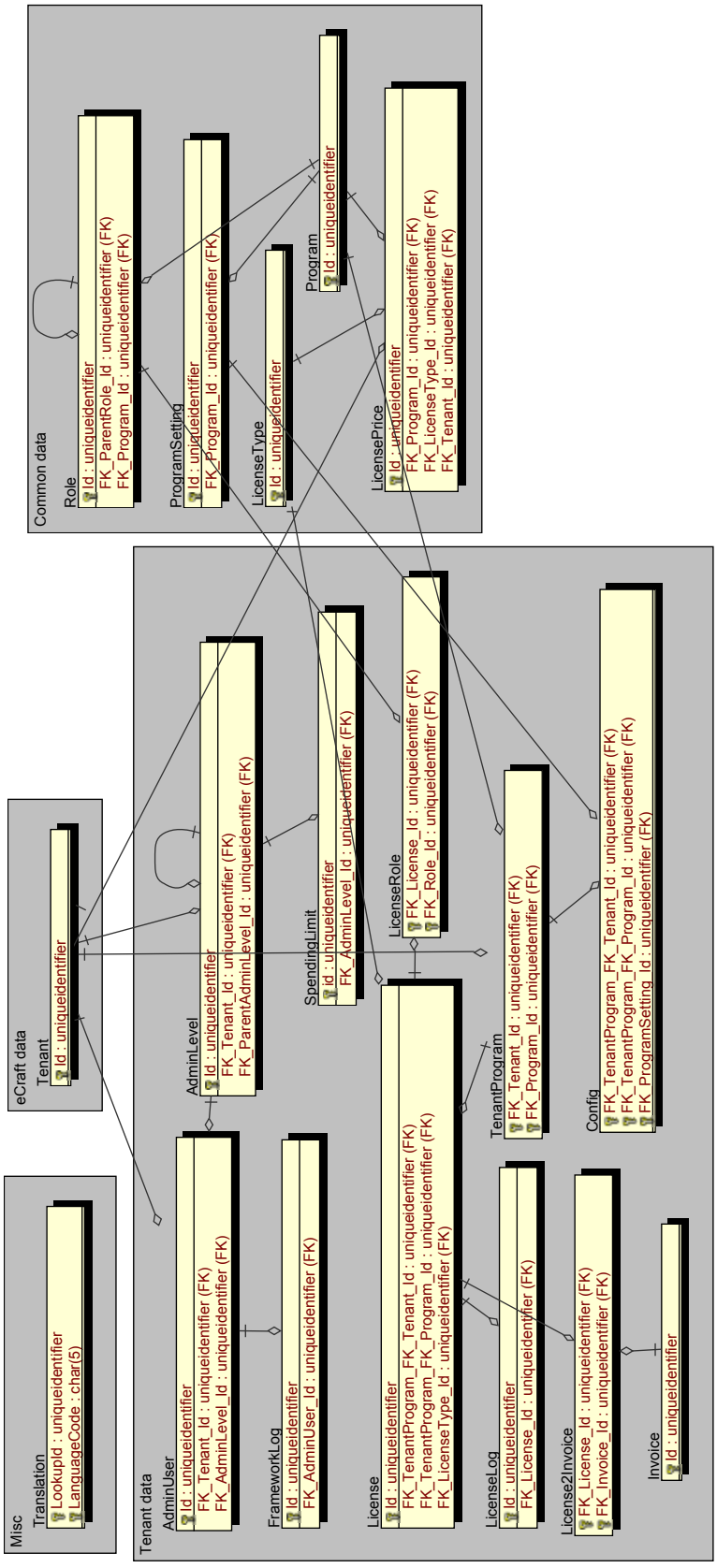

Figure A.1: DB overview

A.2 Common data

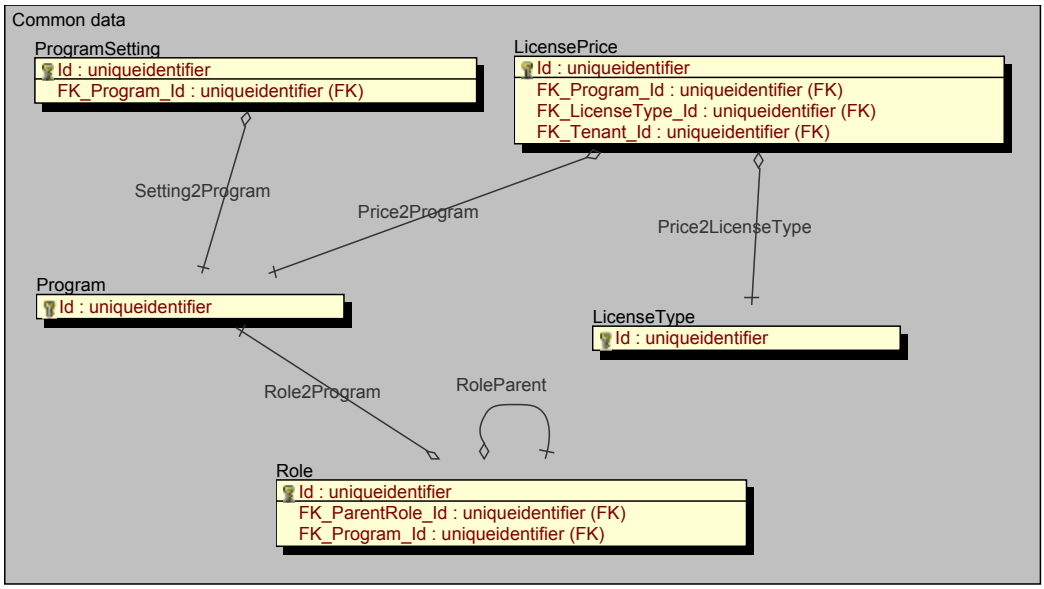

Figure A.2: Common data

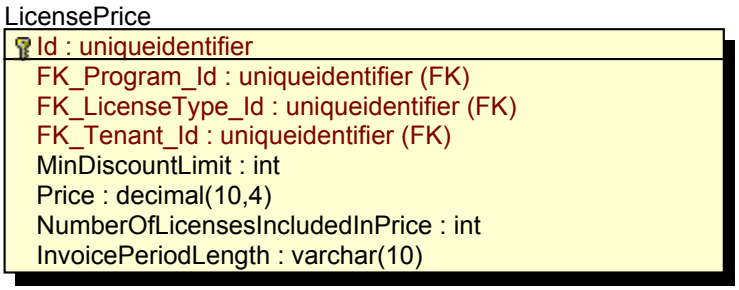

Figure A.3: LicensePrice

Figure A.4: LicenseRole

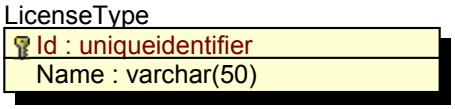

Figure A.5: LicenseType

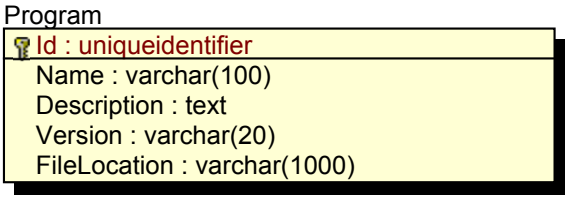

Figure A.6: Program

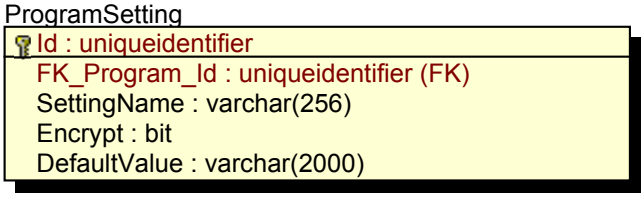

Figure A.7: ProgramSetting

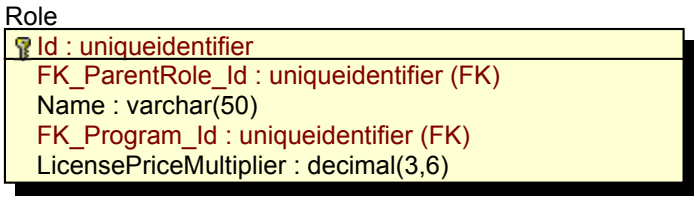

Figure A.8: Role

A.3 eCraft data

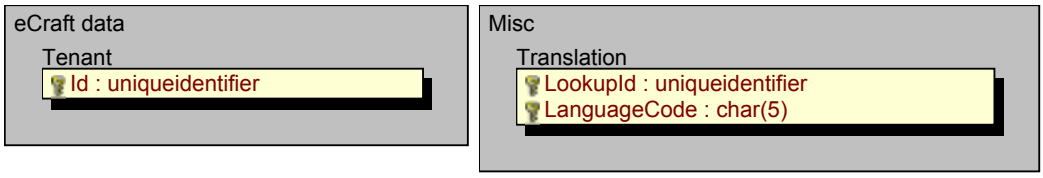

Figure A.9: eCraft data

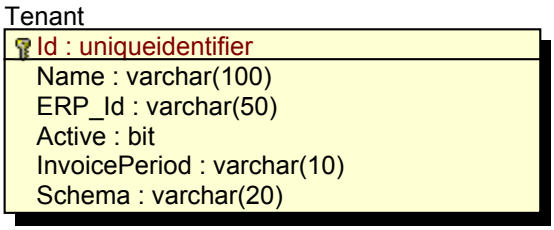

Figure A.10: Tenant

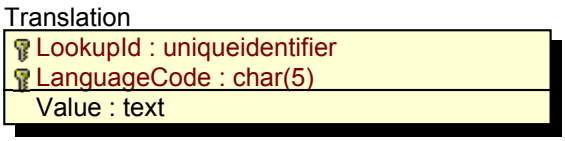

Figure A.11: Translation

A.4 Tenant data

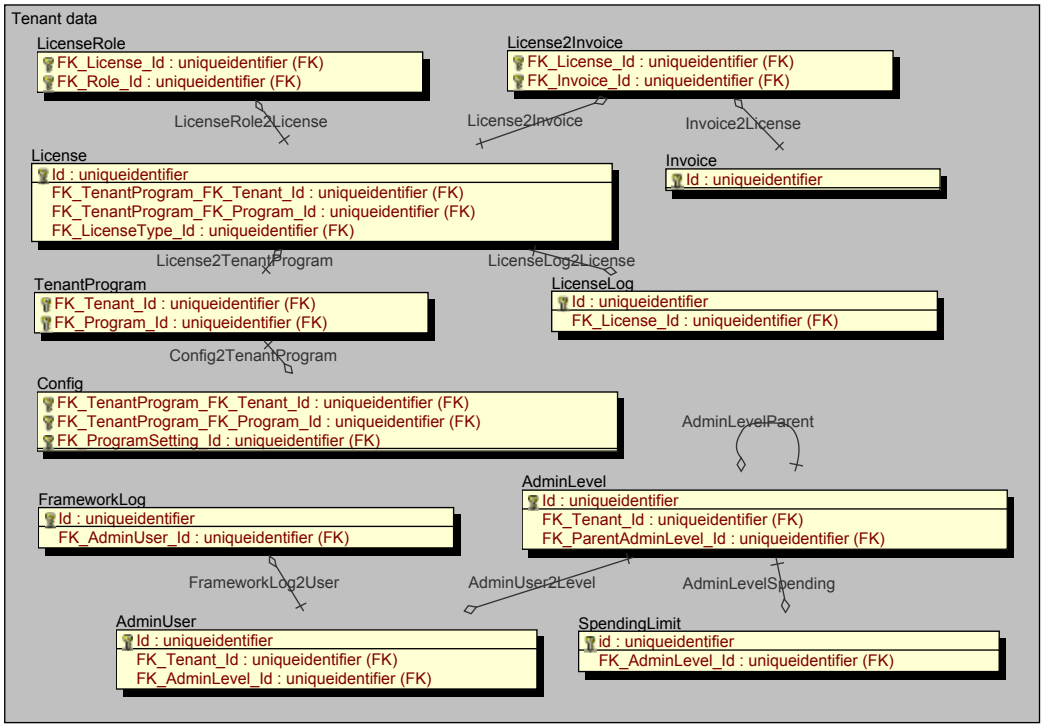

Figure A.12: Tenant data

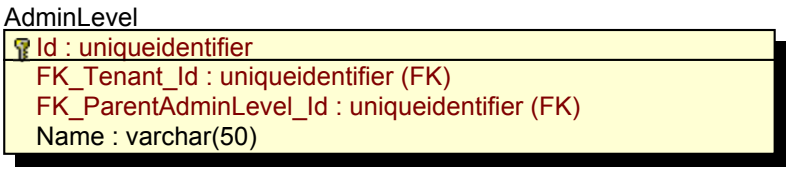

Figure A.13: AdminLevel

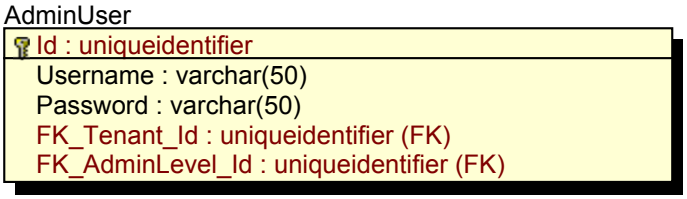

Figure A.14: AdminUser

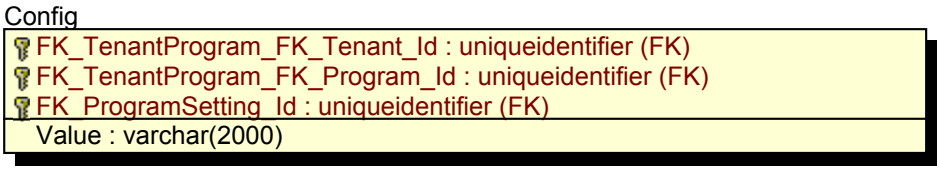

Figure A.15: Config

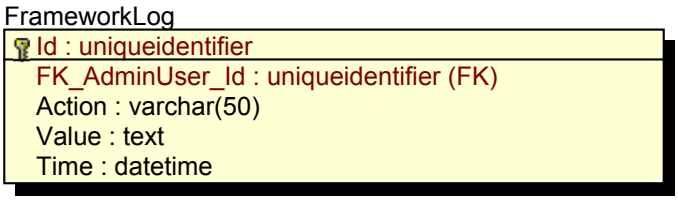

Figure A.16: FrameworkLog

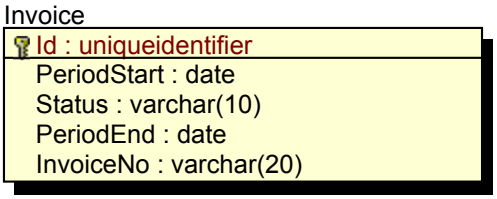

Figure A.17: Invoice

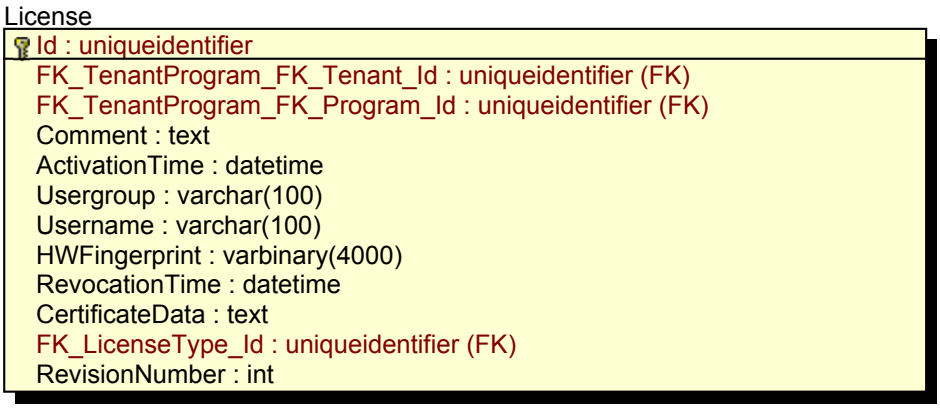

Figure A.18: License

Figure A.19: LicenseInvoice

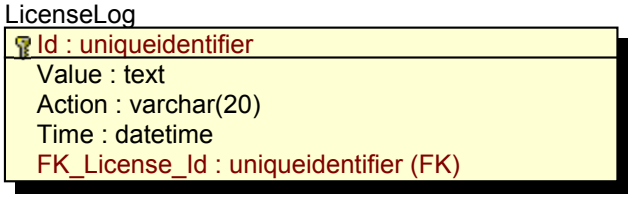

Figure A.20: LicenseLog

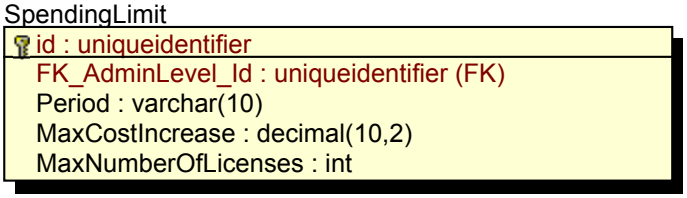

Figure A.21: SpendingLimits

Figure A.22: TenantProgram

Appendix B

License file

```
<?xml version="1.0" encoding="UTF-8" ?>
<license xmlns="http://licensing.ecraft.com/framework/license">
  <licenseVersion>1</licenseVersion>
  <!-- Validity information is updated with each update check. The to value defines when offline use
  is not permitted anymore. A null value means license has to be verified every time the application
  starts. -->
  <validity from="2010-01-01" to="2010-04-01" />
  <softwareInfo>
    <programId>9dcf6530-5427-11df-823b-001bfce2036e</programId>
    <programName>SampleApplication</programName>
    <programVersion>1.0.0.34</programVersion>
    <licenseId>9e824d76-5427-11df-972a-001bfce2036e</licenseId>
    <allowedRoles>
      <role id="">Rolename</Role>
      <role id="">Other</Role>
    </allowedRoles>
  </softwareInfo>
  <clientInfo>
    <!-- depending on license -->
    <hwFingerPrint>0x1234...</hwFingerPrint>
    <userName>TESTDOMAIN\testuser</userName>
  </clientInfo>
  <!-- client signs clientInfo before activating license -->
  <clientSignature>-----BEGIN SIGNATURE----- ...</clientSignature>
  <configuration>
    <licenseServerBaseUri>https://licensing.ecraft.com/</licenseServerBaseUri>
    <connectionStrings>
      <!-- Connection strings should be encrypted -->
      <connectionStringMyExample>
        Data Source=SomeServer;Integrated Security=SSPI;Initial Catalogue=sample_test
      </connectionStringMyExample>
    </connectionStrings>
  </configuration>
  <clientCertificate>-----BEGIN CERTIFICATE----- ...</clientCertificate>
  <signature>-----BEGIN SIGNATURE----- ...</signature>
</license>
```

Listing B.1: Sample license file