SystemWeaver License Manager
A business aware license scheme and implementation

Master of Science Thesis in Networks and Distributed Systems

MIHAI-VASILE BORZ

Department of Computer Science and Engineering
CHALMERSUNIVERSITY OF TECHNOLOGY
Gothenburg, SwedenOctober 2012
Report No.

SystemWeaver License Manager
A business aware license scheme and implementation

MIHAI-VASILE BORZ

Examiner: Olaf Landsiedel

ChalmersUniversity of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Abstract

Over the past decade, the field of programming and computing in general has been marked by the exponential growth of software, web applications and online services. The dynamic of the software industry led to an increase of unauthorized duplication, illegal distribution and use of computer software. Besides, the software companies' constant will of maximizing their revenue, translated into considerable challenges in terms of security and software piracy prevention.

A common method that many IT companies employ for preventing and minimizing the losses due to software piracy is the implementation, besides the actual product, of a license manager application. This one aims to control where and how the software product is used, and prevents its illegal copy and distribution.

The present thesis was conducted at Systemite AB and its purpose was to investigate and implement a license manager for the company's main product, SystemWeaver. For the implementation, the license manager required firstly a rigorous analysis of the company's current licensing model and the security threats to which the software product is subjected to. The design and implementation decisions were based on the company's current needs, tradeoffs between the proposed system architecture and available technologies, possible optimizations and future development. In order to test the validity, functionality and performance of the developed license manager, the system was applied into a simple, practical scenario.

The first part of this thesis presents an overview of the existing software platform that will integrate the license manager application, the company's current licensing model and the threats to which the main product (SystemWeaver) is subjected to. The second part describes the analysis, design and implementation of the license manager starting from the general, conceptual architecture of the system to the detailed definition of its comprising modules.

Acknowledgements

Many people have helped with valuable contributions to my diploma thesis and my sincere gratitude goes out to them.

I would like to thank my supervisor, Magnus Wängefors for his truly motivating involvement and interest in the project, his willingness to spend extend periods of his time researching possible approaches in such a complex domain. During our project meetings he offered continual guidance, expert knowledge and assistance in a highly enthusiastic and positive manner.

I would also like to thank the Olaf Landsiedel, for providing me with key pointers on the direction of the project and suggesting different approaches.

Their advice throughout the project development process has been firm, dependable and constructive.

Contents

List of figures

List of tables

Chapter 1

Introduction

This chapter makes a short introduction to the subject area of the master thesis. The current context, a brief presentation of the existing software platform that will integrate the license manager product together with the main fundamental concepts behind it, cover the first part of this chapter. Further on, a brief presentation of the existing software platform that will integrate the license manager product is made. We also state few fundamental concepts behind such an application and, toward the end of the chapter, we refer to the general domain of the problem and state the approach used for documenting and developing the project.

1.1 Project Context

The research and development of this master thesis was conducted at Systemite AB, based in Gothenburg, Sweden. Systemite provides a high performance open platform for component-based systems development within the area of computer based systems. The research and development of this master thesis was conducted at Systemite AB, based in Gothenburg, Sweden. Systemite provides a high performance open platform for component-based systems development within the area of computer based systems. "The platform, named SystemWeaver, enables engineering organizations to integrate their design processes using one enterprise-wide data repository that assembles the design information of each component, such as: internal structure, interfaces, variants, versions, requirements, status etc." [1]. In other words, the platform contributes to enhancing product quality, gives the possibility of refactoring and rationalizing the entire design process, provides better information maintenance mechanisms and, nevertheless, offers improved methods for organizing, planning and managing projects.

At this point, SystemWeaver lacks a license management tool able to prevent potential losses that may occur due to unauthorized replication and use of the application.The term “License management” is a complex one and it mainly refers to the ability of tacking and generating licenses for customers. Nevertheless, a license management tool can also control where and how a software product (in this case, SystemWeaver) is able to run, hence being a valuable asset in preventing losses due to software piracy.

It is important to differentiate between license manager and software asset management tools. While the former has the purpose mentioned above, the latter is used to manage, within a company or organization, the software licensed from other vendors. We underline here that the outcome of the present master thesis project - SystemWeaver license manager -, is intended to be a custom/private license manager implementation that will meet Systemite's and its customers' needs.

1.2 Problem Domain

A proper solution for the license manager presented above requires studying different license validation procedures, Systemite's licensing models, different cryptographic algorithms, technologies and related work. Based on the company's sales, development and deployment process, as well as on customers' process and the way they are using the software product, tradeoffs between security and availability must be identified. Last, but not least, one must take into consideration the involved costs, maintainability and extensibility of such a product.

Given the considerations above, we approached the project in the following manner: a first step was to gather stakeholder's needs and to investigate the company's current workflow. Further on, it was analyzed how the customers are using SystemWeaver and what are the security threats to which System Weaver is exposed. The obtained results were integrated in a Vision document together with the non-functional requirements.

A second step aimed at proposing and validating a suitable solution. It involved analyzing the requirements and proposing several methods that will fulfill them. The proposed solutions were validated together with the stakeholders and the most suitable one was chosen. The last step involved implementing the validated solution at the previous step.

In general, providing any software product with a license manager holds numerous advantages that might outweigh the initial efforts and costs required by its development. For example, a license manager can strengthen the company's and the product's brand image among the existing customers, consolidating its position and credibility on the development process oriented software market. From a financial point of view, such an extension opens the door to new customers and sales perspectives on the international market and generates incremental revenue beyond the regular profit stream.

Chapter 2

Project theme and objectives

The current chapter presents the theme of the master thesis and its tangible objectives. Each goal is detailed so as to show the tasks involved and some of the related challenges. The chapter concludes with an overview of the rest of the document, a short presentation of chapters which follow.

2.1 Problem statement

The licensing management and strategies are no longer disregarded in today's ever changing IT climate. The fast evolution of the IT market together with the financial and operation risks to which all independent software vendors are exposed to (e.g. lack of control over the availability and/or replication of the software), determined them to focus more and more on solutions that are beneficial for their long term progress and revenue. As a company grows towards an enterprise, the license management becomes a bottleneck in company's operations. Therefore, more and more effort is required for developing a license extension to the actual software product.

License manager is as well a valuable asset in preventing losses due to software piracy. Also, given that SystemWeaver customers are increasing constantly in number and in geographic areas, it becomes more and more difficult to keep track of the expiration date, the number of users that are using the system, hence making software replication becomes possible.

In this context, a license manager for SystemWeaver becomes a must. The sales department is encountering difficulties and is spending unnecessary overtime counting users, keeping track of their software license expiry date and informing the customers to renew their license agreement. This results in delays and potential business losses.

2.2 Objectives

This section presents the objectives and the scope of the License manager system in a structured manner, covering the theoretical aspects, the design issues and the experimental implementation goals.

Throughout the project development the following key ideas served as guiding pointers:

- Collect, analyze, and define high-level needs and features of the SystemWeaver license manager. It focuses on the capabilities needed by the stakeholders and the target users, and why these needs exist;
- Investigate both Systemite's and its customers' current workflow and the security threats they are exposed to;
- Identify the key requirements related to license generation and management and define the needed security level;
- Define a flexible solution that is able to accept changes to the workflows and that meets the stakeholders needs;
- Implement a prototype as a proof of concept for the proposed solution.

2.3 Challenges

The final product should not affect Systemite's current processes of development, sales and deploying. Also, the cost of maintenance and extensibility should not be prone to an increase due to the proposed solution or its future development. Although the license tool should prevent the unauthorized use of System Weaver product, it should still provide high availability to Systemite's customers.

To summarize, the main challenges are:

- Mediate between different contradictory needs;
- Define the tradeoffs between security and availability, based on the stockholder's needs and choose the appropriate software license model;
- Evaluate the security level that is necessary or suited;
- Analyze available cryptographic algorithms and choose an appropriate one to be used in the actual implementation.

2.4 Report organization

This first chapter of the paper consisted in a short introduction to the subject of licensing manager in the context of a software product. The current section, Chapter 2, details the objectives of the project, the systematic task list and highlights some of the thesis challenges.

The third chapter makes a review of the literature studied throughout the development of the application. It investigates different lines related to licensing schemes in enterprise computing scenarios and briefly describes several cryptographic algorithms and block ciphers' operation modes. Towards the end, the chapter covers a short introduction to SystemWaver collaborative environment.

Chapter 4 presents some available license manager solutions existing on the market, namely Microsoft Office and Hydra.

Chapter 5 lays out the fundamental aspects: the analysis and design of license manager. Here is explained the general system architecture and its modules. It is also shown the roles of each module from a conceptual point of view and the way these modules interact. The focus here falls on the decisions and existing tradeoffs based on the needs of stakeholders and their customers.

In Chapter 6 is discussed the development and the implementation of the proposed solution. Also, more details are offered regarding the inner-workings of the modules introduced in the previous section. Relevant details about the employed technologies are explained and the choices we made are also motivated here.

Chapter 7 concentrates on the methods used to test the system, in order to confirm its validity. We apply the system in a simple practical scenario and we investigate all the functional requirements. The system's performance is also tested. We investigate the overhead that is added by the license manager to the authentication process.

Finally, Chapter 8 contains the conclusions of this thesis and gives future development directions. The conclusions drawn after analyzing, designing, implementing and testing the license manager are presented, by underlining the advantages of the selected solution.

Chapter 3

Literature review

This chapter makes a review of what was done and studied in the area of license management and license validation procedures. Since the topic is vast and diverse, the review is by no means exhaustive, but it should provide relevant hints on the issues related to license management and product validation. The chapter begins by presenting some license validation procedures. We continue with describing the available cryptographic algorithms and their features in order to understand the designing decisions presented in the following chapters.

For a better understanding of the subsequent design and implementation decisions, we present a high level overview of Systemite's product (SystemWeaver), how this one works and how it meets the customer's needs. Finally we motivate the need for a proprietary solution, we present the challenges it address and try to position the approach adopted for the present work in the outlined context.

3.1 License validation procedures

In the proprietary software industry, every software application is accompanied by a licensing agreement that establishes the rights the purchaser has for using that software. Such an agreement defines the terms under which the purchased software copy can be used and, commonly, is contained only in digital form. The agreement is most of the time presented to the user as a 'click-through' procedure that he/she must accept.

Hence, the term 'licensing validation' stands for the procedure carried out each time a user is entering the software product licensing information. It is a procedure used to verify that the software license in use is in accordance with the End User License Agreement.

3.1.2 Product activation

Product activation is a license validation procedure required usually after the installation of software programs. Its main purpose is to reduce a form of software piracy known as "casual copying" or "soft lifting" and ensures that end users using the final product will receive the product quality that they are expecting.

Casual copying is a form of software piracy in which users share the software in a way that violates the software license terms. For example, a user buys an operating system for a single computer, but he installs it on an additional computer. Another example of casual copying is when a user buys a media format (most of the media formats are easy to replicate) and creates copies that he shares with others. Casual copying accounts for a large part of the piracy losses that the software industry experiences.

3.1.2.1 How product activation works

Product activation is complete software and does not need neither special hardware nor any other external tools. The software vendors usually send to the user a unique product serial number. While installing the application the user has to type in the product identifier. A unique identifier for the particular machine on which the product will be running is created by the product activation software by simply hashing the hardware serial numbers. Further on, both the machine and the unique serial number are sent via Internet to the manufacturer that verifies their validity and whether they were used for multiple installations.

The application will receive license information from the manufacturer describing the user’s license like time limit, list of features available to the current users and so on. After the activation, the user is allowed to use the product based on the time limit. In case the customer needs extra features or has to purchase an additional license, the activation procedure is repeated.

3.1.2.2 Drawbacks

The product activation has as well disadvantages that are presented in the following paragraphs.

The first disadvantage is the high cost of implementation. The software vendors need to have a product activations center that has to provide high availability to the end uses. The product activations servers should be always up and running or, in the case of telephone-based activation, there is a need of an automated telephone system or customer responsible personel.

Another disadvantage is that there are situations where the target machine does not have an Internet connection, thus forcing software vendors to implement a telephone activation system. However, some activations systems support activations without Internet or telephone connections. A common approach is to exchange encrypted files.

3.1.2 Other validation methods

There are no standard ways of validating a license. Software vendors usually build their own custom methods. There are vendors that are sending license files that are activating the features or use key generators and compare if the introduced product key was generated by the product key generators. However, there is no standard technique and usually several techniques are combined for validating a software product.

3.2 Encryption algorithms

All the algorithms presented in this paper are symmetric algorithms.

3.2.1 Symmetric key encryption

Symmetric key encryption (also known as conventional encryption or single key encryption) is still a widely used cryptography model and, in some case, it is employed together with public key encryption. In symmetric key encryption, the process of encryption uses the same key as the decryption process i.e. the decryption algorithm is the reverse of the encryption algorithm. More specifically, the encryption process consists of dividing the plain text into several small blocks/chunks (64, 128, 256 bits); the blocks, together with a secret key, serve as input to the encryption algorithm that performs different transformations and substitutions, hence resulting ciphertext blocks.

A symmetric algorithm is usually faster than a public key one and ensures the same security level but with a smaller key size. On the other hand, it holds some disadvantages such as: in order to decipher the text, the key must be sent through a secure channel. Also, the fact that several blocks are encrypted using the same key, raises multiple security issues and makes this type of encryption algorithm prone to brute-force attacks and cryptanalysis (based on algorithm's properties).

3.2.2 Modes of operation

NIST (National Institute of Standards and Technology) defined five modes of operation for the block ciphers. Essentially, it is underlined not only that the block ciphers may be used in numerous applications but also that the cryptographic algorithm may be adapted to the application in order to enhance its effect.

3.2.2.1 Electronic Codebook mode (ECB)

Electronic codebook (Figure 3.1) is the simplest mode of operation, in which the plaintext is split into equal sized blocks (last block may be padded, if necessary) and each block is encoded independently using the same key. Thus, to each plaintext block corresponds a cipher text block and, as a characteristic (and also disadvantage) of this mode of operation - a plaintext block appearing repeatedly in the message will produce the same ciphertext – see Figure 3.3. In conclusion, ECB may be suitable for encrypting rather short amounts of data, but for longer messages that contain repetitive sequences or that are highly structured, ECB is no longer a secure alternative (Figure 3.3).

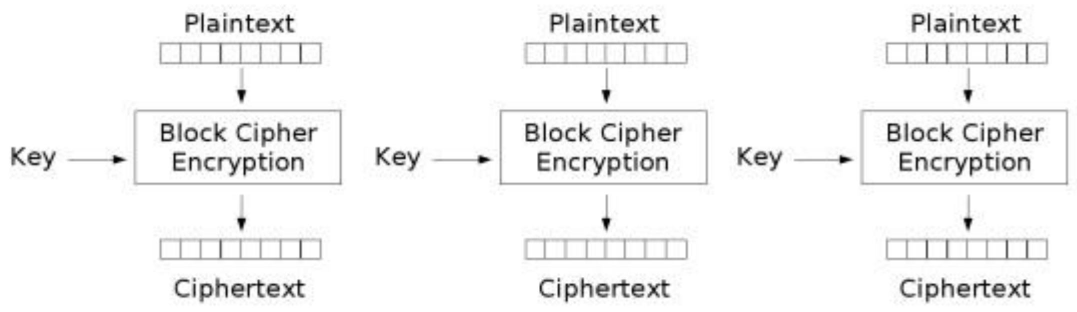


Figure 3.1 Electronic Codebook (ECB) encryption [2]

3.2.2.2 Cipher –Block Chaining mode (CBC)

As it can be seen in Figure 3.3, cipher block chaining mode (CBC) overcomes the drawbacks of ECB, thus offering an increased security level. In the CBC mode, the input to the encryption algorithm is given by the plaintext block combined – XORed - with the ciphertext resulted from the previous block.

Note that the encryption of the first block is done using an initialization vector, sent in plain text at the beginning of the encryption procedure. Although the key for each block is the same, the encryption of subsequent plaintext blocks is chained with the processing of present/previous block, hence ensuring that the encryption of repetitive patterns will result in different ciphertext blocks. Given its chaining mechanism, CBC proves appropriate for encrypting lengthier messages and it is used not only for achieving confidentiality but also for authentication purposes.

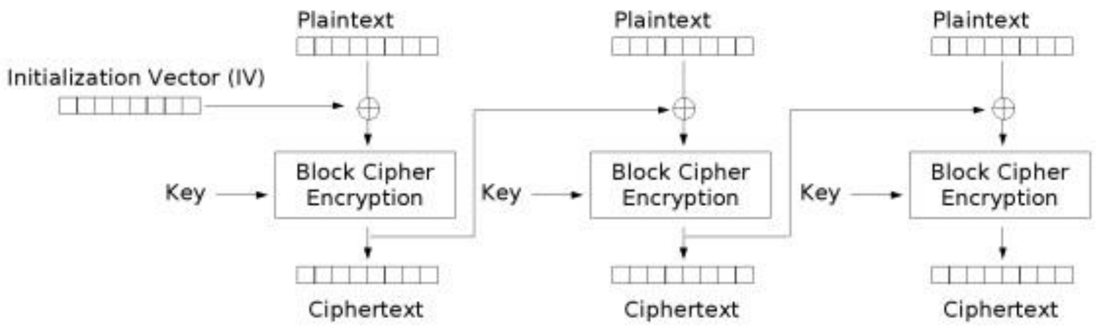


Figure 3.2 Cipher Block Chaining mode encryption (CBC) [3]

Note that in the case of both EBC and CBC, when partitioning a lengthy plaintext into equal blocks, the last block may require to be padded in order to acquire a certain size. To eliminate the need for padding and also to enable also the cipher to operate in real-time mode, block ciphers may be converted into stream ciphers, using the operation modes described in the followings.

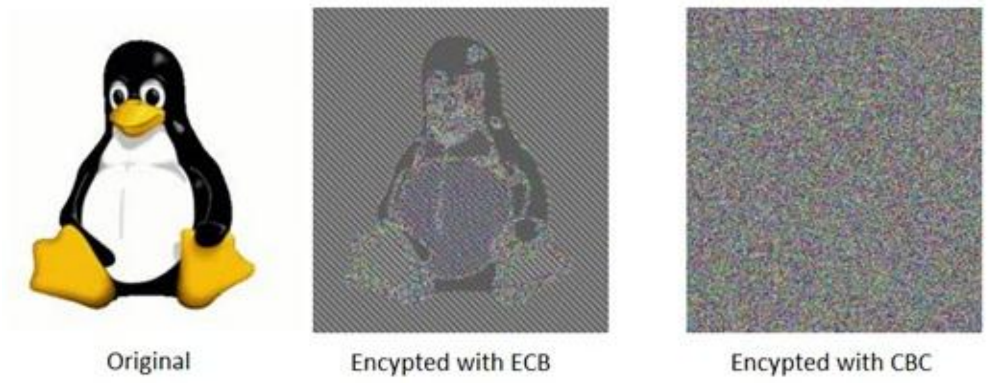


Figure 3.3 ECB vs. CBC [4]

3.2.2.3 Cipher Feedback mode (CFB)

Although it is not properly conformed to the construction of a stream cipher, CFB may be viewed as one. Just like the previous mode, CFB operates also in a chaining mode. Hence, the input to the encryption block is a register (initially set to an initialization vector). The most significant bits resulted after the encryption are XORed with the first plaintext block in order to produce the first ciphertext block. Additionally, the contents of the shift register are shifted left (by a number of bits equal to the length of the ciphertext) and the resulted ciphertext is placed in the rightmost bits of the shift register. The process is repeated until all plaintext block are encrypted. Just like in CBC mode, the initialization vector may be sent in clear text but it should be different for messages encrypted with the same key.

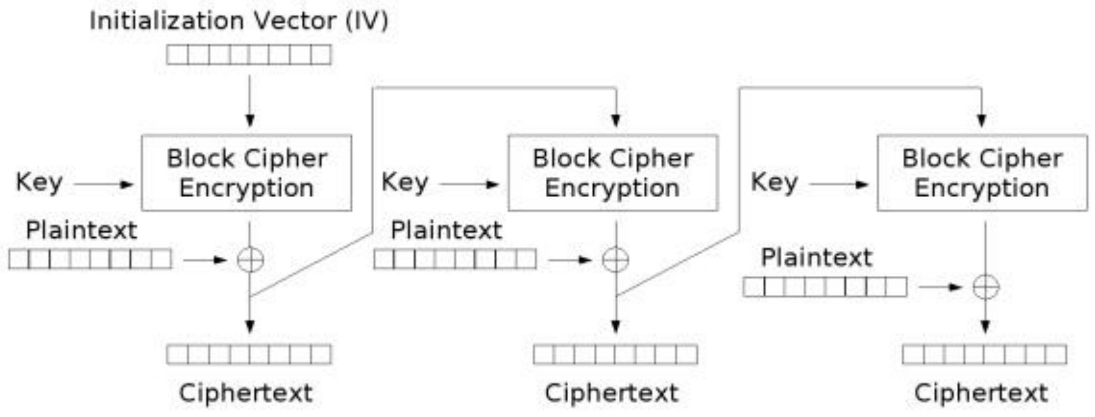


Figure 3.4 Cipher Feedback (CFB) mode encryption [5]

3.2.2.4 Output Feedback mode (OFB)

Unlike CFB, where the resulted ciphertext is fed back to the shift register, in the case of OFB is the output of the encryption block that is input to the shift register. This makes OFB structurally similar to CFB but, security-wise, more vulnerable to message stream modification attacks (e.g. a change in the ciphertext is reflected in the decrypted text, thus the integrity of the message can be easily affected) and less prone to propagating transmission errors.

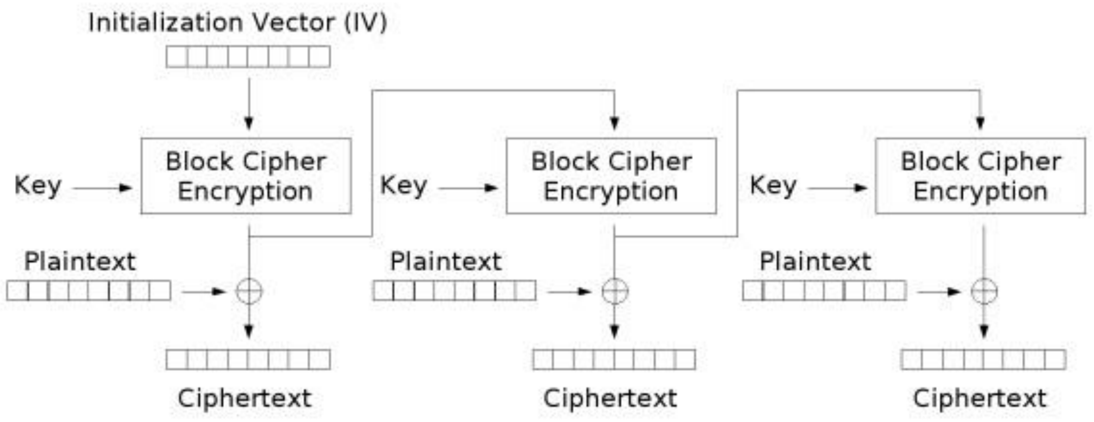


Figure 3.5 Output Feedback (OFB) mode encryption [6]

3.2.2.5 Counter mode (CTR)

Counter mode (Figure 3.6), just like cipher block chaining, combines the plain text block with the output of the counter. Combining the counters output with the plaintext ensures that similar blocks of the plain text are encrypted separately. However, in counter mode, blocks do not depend on each other. This means that even though a malicious person can change a block, the other blocks of the plaintext will not be affected. In the CBC mode changing one block will create an avalanche effect that affects all the following blocks that follow the affected block.

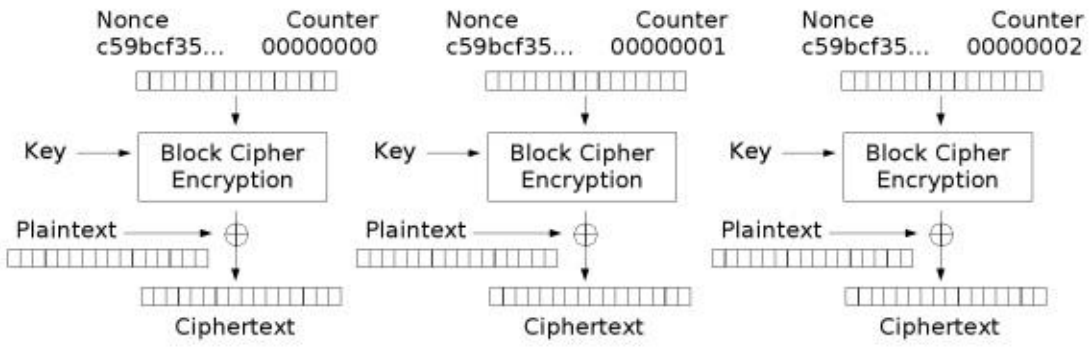


Figure 3.6 Counter mode encryption [7]

3.2.3 AES and DES

Data Encryption Standard (DES) was one of the most used encryption scheme. The algorithm takes as input 64 bit blocks and encrypts them using a 64 bit key (more specifically only 56 of these bits are used, the other 8 being left as parity bits or set arbitrarily). For a detailed description of this algorithm refer to [8]. Note that technological advances in hardware and parallel computing made DES insecure and in order to improve its security, the key size was doubled or tripled while applying the algorithm twice respectively three times. Although the key space increased (twice or three times), the outcome was not as expected, the security level failing to expose a double or triple increase. Moreover, the new algorithm (3DES) became too expensive in terms of computational resources.

As a replacement for DES, a new block symmetric block cipher was approved for a wide range of applications: AES. Unlike other block ciphers, AES's structure is rather complex (for its complete structure and description, see [8]), but its main advantage is a higher speed on a wide range of CPUs i.e. from 2010 Intel processors include AES-NI instruction which perform AES operation in hardware.

3.3System Weaver

SystemWeaver is a Systemite product. SystemWeaver is a distributed environment which provides to its customers real time collaborations between globally distributed sites. Basically, all SystemWeaver users have instant access to all product data sharing and building on each other's result in real time. SystemWeaver allows different development disciplines and processes.

The SystemWeaver platform uses custom IT solutions mixed with proprietary technologies all together aiming to reach a high performance. Moreover, it is a general solution that can be customized for each customer without the intervention of the core development department based on the idea of model based development. The customer products and workflow are modeled with a metamodel by the Systemite's application engineers.

SystemWeaver environment is formed of: SystemWeaver Model Server, SystemWeaver Mirror Server and SystemWeaver Client. SystemWeaver Model Server is the main server that coordinates all other servers. It accepts as well connections directly from the SystemWeaver clients. This server contains the metamodel that describes the customers' products and workflow as well as authentication information and users' data.

SystemWeaver Mirror Server is a proxy on each client site. It mediates the connections between SystemWeaver client and SystemWeaver Model Server. It also chases data to avoid as much as possible unnecessary communication. Each time the data has changed the SystemWeaver Model Server sends change events to the mirror servers. In order to keep providing real time collaboration, the mirror servers use a write through policy i.e. in the case of a writes in the cache and send further the write to the SystemWeaver Model Server. Due to the high amount of work, SystemWeaver Model Server is kept as thin as possible. Most of the business logic is implemented on the client side.

Note that there are several types of client applications, each being designed depending on the aimed functionality and end users. The first client application - SystemWeaver Admin – is designed for system administrators and its purpose is to offer CRUD functionality for users, addition of corresponding roles for its entities and the assignment of user permissions. The second client application –SystemWeaver Architect- is aimed for metamodeling and its final users are application engineers. The third client application –SystemWeaver Explorer- is aimed for the end users and based on the metamodel it interprets the clients data.

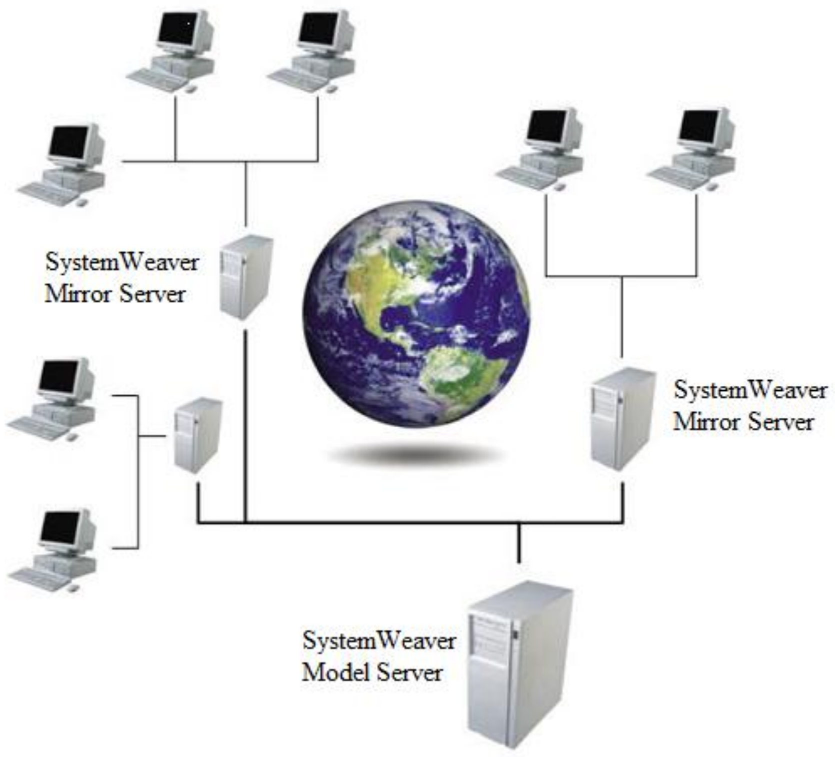

SystemWeaver Mirror Server
SystemWeaver Mirror Server
SystemWeaver Model Server

Figure 3.7 SystemWeaver distributed architecture

Chapter 4

Related work

Given the fact that at present time there is no standard validation procedure for the software products, the market offers a wide selection of license manager solutions. In the followings are briefly presented some examples implemented by other software vendors.

For example, Microsoft started using product activations in Microsoft Word 97 soled on Hungarian market. Microsoft supports two ways of product activation: over the Internet or over the telephone. Over the Internet, the Microsoft servers process the activations requests and activate the product. For the telephone based activations one has to call the Microsoft product activation center and must follow the steps provided by an automated telephone system or by a customer representative.

Another example is “RemObjects Hydra - an application framework that allows developers to create modular applications that can mix managed (.NET) and unmanaged (native Delphi) code in the same project, creating a seamless user experience while combining the best technologies available from either platform” [9].

They provide users with a trial version of the framework without any technical limitations but tools, compilers and .NET libraries expire 30 days after installation. The project compiled with the trial version will produce a message dialog indicating that it was created with a trial version of Hydra. In this case, the vendors are selling licenses per developer. For buying a license, the user must firstly be registered on RemObject’s web site. Once the license has been purchased, the user receives a license file that must be registered. During the registration, the user is required to enter his credentials ensuring in this manner that only one copy of the license file can be activated.

Chapter 5

System Analysis

This chapter presents the details regarding the analysis of the License Manager. The chapter begins with a presentation of Systemite's licensing model and high level needs. Afterwards we sum up some of the most important requirements that deal with security and which had a large impact on our design decisions.

5.1 Systemite's licensing model

Today the company sells both temporary and permanent licenses of SystemWeaver. The licenses are sold per user and, given the fact that the system supports different types of end users – namely viewers (that have only read access capabilities) and regular users –, different charges are applied.

Currently, there is no mechanism to prevent the addition of new users into the system and usage after license expiration time. Basically, once the product is sold to a client, this may take full advantage of its features for an unlimited number of users. Besides, the sales department encounters difficulties and spends unnecessary overtime counting users or informing customers that their license has expired and that they have to renew the license agreement. This results in delays and potential business losses.

5.2 High level needs

In this section we define high-level needs and features of the SystemWeaver License Manager. The section focuses on the capabilities needed by the company's employees and the target users, and **why** these needs exist. The details of how the sales department fulfils these needs are detailed in the following sections.

5.2.1 Systemite general needs

5.2.1.1 Usage limitation

One of the main needs is to limit and prevent piracy and unauthorized usage of SystemWeaver system. Once the license period expired, the customers should not be any longer entitled to use the software. Note that at present there is neither a demo version and any other kind of usage limitation, nor a certain trial period under which a customer can use the software; hence, once someone obtains a copy of the application, he will be able to use it freely, without any constraints. On long term this brings high financial and business losses to any software vendor.

Under these circumstances, the aim is to have a license agreement between the software provider and the customer in which is stipulated the exact number of users that will be using the system. From a technical point of view, the license manager application should be able to limit the user accounts number to the one agreed with the customer.

5.2.1.2 Overhead in developing and usage process

The License Manager should not add overhead to Systemite's departments and customers. In contrast, introducing the license manager to the actual software package should optimize and minimize the administrative overhead with which the sales department is confronting. Also, the clients (system administrators) will either receive, on timely basis, the status information regarding the purchased licenses and the expiration time, or will be able to access by themselves this information. Warning the users regarding the license status is intended to be an unintrusive process that will affect by no means the customer's normal workflow.

5.2.2 Sales department

5.2.2.1 Clients management

Another requirement for the licence manager is to store client general information and support CRUD (**C**reate/**R**ead/**U**pdate/**D**elete) functionality for these entities. This includes also address information and contacts. Licence management should be able to accommodate tracking of licences, invoicing and maybe integration with a financial system.

5.2.2.2 Users and system administrators alerts

The SystemWeaver users and administrators should be informed when they are close to (or reached) the expiration date or when they approached (or reached) the maximum amount of resources agreed in the license.

5.2.2.3 Deactivate the license manager for trusted users

The license manager is not intended to be an impediment or to bring any prejudices to the regular workflow of trusted customers. Systemite is willing to provide software availability to trusted customers even after license expiration.

5.2.3 Product (core) development department

5.2.3.1 Costs

License manager should not increase the costs of maintenance and future development.

5.2.3.2 Server migration

If the customers need to move the server to new hardware they should not need to contact Systemite. For example, in case of a server machine failure, in order to provide high availability the SystemWeaver server must be migrated quickly to a different machine.

Some clients develop their own metamodel and have the QA department testing it. Therefore, the QA needs a copy of the current database in order not to affect or alter the developers' real data.

5.2.3.3 Active license manager

The license manager should be always active i.e. all builds should include the license manager. Since it is cumbersome to manage all the builds, it is very easy to leak a build without a licence manager.

5.3 Desired security

In order to define the security level of an application, one has to analyze security treats, has to take into consideration the type of users and customers the application is intended forand, nevertheless, the areas of usage and the number of users.

5.3.1 SystemWeaver vulnerabilities and treats

A vulnerability of SystemWeaver is due to the fact that Systemite sells licenses per user and/or usage time. After deploying the system, the company has no control over the number of users that are using the application. At the end of the agreed time period, when the licenses must be renewed, the financial department will start counting the current number of active users. However, when the time to negotiate a new contract comes, the customer's system administrator can deactivate users and activate them again after renegotiations. Currently, customers are still able to use SystemWeaver even after license expiration, therefore Systemite sales' department has to contact them and initiate the negotiations.

Another threat is that SystemWeaver is easy to replicate and reuse. The applications contained within SystemWeaver are simple executable files and no installation wizard or validation is required. Therefore, one can get a copy of SystemWeaver and use it without any impediments.

5.3.2 SystemWeaver customers and area of use

Taking into consideration that SystemWeaver is a system designed to provide precision and power in managing complex business models, it is used mostly in the industry and in organizations that are geographically distributed, where a constant need of collaboration between engineers exists. Therefore, SystemWeaver is useless for domestic use and the final customers are mostly in the high-tech industries (automotive, aerospace, construction equipment) such as Volvo Group, Denso, Ford Motor Company, Land Rover etc.

5.3.4 Security requirements

After analyzing the overall requirements, in this section we extracted and redefined those that are directly related to security. These are the followings:

- In case someone holds a copy of SystemWeaver applications he/she shall not be able to take advantage of SystemWeaver's full functionality in the absence of a license. Hence, there is a need of a trial mode in which the application has resources constraints or features limitation.
- After the expiration date, it shall be impossible to use SystemWeaver unless the current resources are downgraded to those available in the trial version.
- One should not be able to add more users than the ones agreed in the license.
- The license manager shall work offline. There are customer sites where the machine that runs the SystemWeaver server does not have an Internet connection.

5.3.5 Attacker model

Our focus is not on SystemWeaver's security but on the license manager's security. Therefore, an attacker can take advantage of any methods except reverse engineering the SystemWeaver server. Reverse engineering the SystemWeaver server is an issue related to the SystemWeaver's security and is not the purpose of this project.

In this section we try to propose a well defined attacker model for the license manager. Defining the attacker model is of a big importance since it helps the designers to rigorously analyze the security threats the application is subjected to. Moreover, it eases the process of comparing various solutions against each other and selecting thesuitable one.

According to [10], an attacker model comprises of a set of basic attacker models. At its turn, a basic attacker model can be defined as a pair (i, p) of values, where i stands for "Intervention – What the attacker can do" and p stands for "Presence – Where can he do it".For a correct and realistic identification of the (i,p) pairs that map onto our system, we focused mainly on how an attacker can modify, affect or disrupt the normal behavior of the application. Nevertheless, from where the attack may be conducted and what are the points and parts of the system that might be affected by an attack must also be taken into account.

Intervention:

- Enable extra resources in trial mode: when running in trial mode, an attacker should not be able to enable/use more resources than those allocated for trial mode;
- Addition of extra resources (in our case, user accounts) over those stipulated in the license agreement: although similar to the previous one, this intervention is possible only after license activation;

- Usage of the same number of resources after license expiration;
- Use the License Manger to perform a DoS (Denial of Service) attack;
- Affect time's flow on the server's side (make the time run backwards at server's side). This intervention affects the expiration date. More specifically, an attacker can postpone to an indefinite date the license's expiration by simply making the time on server's side run backwards;
- Eavesdropping – this intervention is more related to SystemWeaver's security. However, it can also be regarded as a license manager problem since an attacker can eavesdrop the license registration messages and reuse their content.

Presence:

- **Local**: the attacker has direct access to the machine that runs the SystemWeaver server and also to the data base;
- **Remote**: the attacker can access the server through the client applications.

5.3.6 Security level analysis

According to the presented vulnerabilities and risks, but also considering the company's current size, its actual customers and the area of use, there is no need for an expensive solution. However, the solution should be resistant to the above mentioned security requirements and attacker model.

Moreover, it is worth investing and developing a highly secured mechanism only for those products that can be used for personal purposes. A trivial example would be MS Office Excel that can be used by companies but also by single users. Hence, the rate of attempts to illegally use and replicate this software is higher than for a product that has no use for a regular user. As stated previously, SystemWeaver is not aimed for domestic use therefore it is worth finding tradeoffs between security and costs.

Chapter 6

System design

We begin this chapter by introducing the main idea on which the system's architecture relies on. In the last part we focus on a more detailed description of the system's conceptual architecture and design.

6.1 Basic idea

We chose to use as license validation the file based approach, where customers receive a license file that activates the resources they paid for. In our case, the license file will mainly contain information about the number of users that are allowed to use the application, the time duration they are allowed to use it, as well as configuration data that describes how the system should behaving under certain conditions.

Given that Systemite is a small company and does not have enough resources to afford product activation, the license file approach is preferred to the product activation one. Another reason behind our option is that SystemWeaver is not intended for personal use -it is an expensive system designed for large enterprises – therefore, the number of customers is not that high. Moreover, most of Systemite's customers keep confidential data into SystemWeaver server, making communication with a product activation server difficult. The telephone activation is excluded since the implementation costs are too high.

Using a license file, the design is flexible and it is easy to change to product activation in the future. Meanwhile, the license file can be used as well for disabling and enabling different product features that are customizations for particular clients. Also, an Internet connection is not mandatory. On the other hand, this approach has the disadvantage that a license file may be used to register multiple copies of the database.

6.2 System conceptual architecture

The conceptual architecture of the license manger as well as the involved components and the interaction between them is presented in Figure 6.1.

OnSystemite's site there is a standalone application, License management tool, responsible for license generation. The tool is used also by the sales department to keep track of the customers and sold licenses.

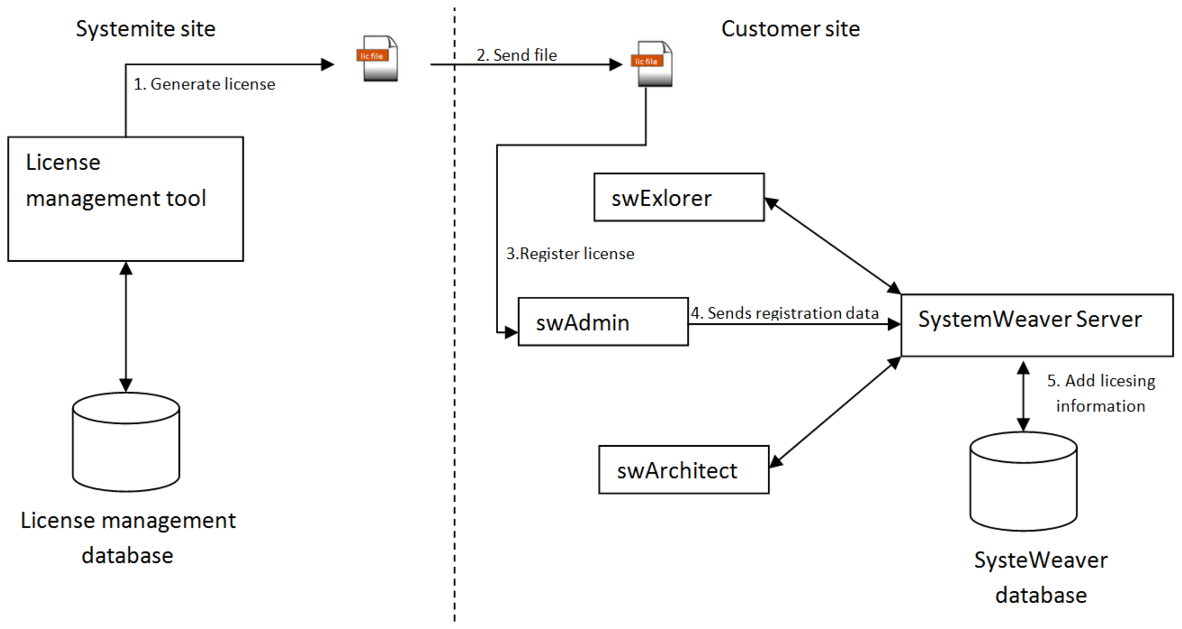


Figure 6.1 Component architecture

As it can be seen in the Figure 6.1 only **swAdmin** tool and the **SystemWeaver** server need to be modified. The **swAdmin** tool needs a mechanism for registering the license file while the **SystemWeaver** server needs a mechanism for preventing unauthorized use and for registering the license file. Each component will be detailed in the detailed design section (Chapter 6.3).

Figure 6.1 shows also the interaction between the components. Therefore, the normal usage flow of the system is as follows:

1. A standalone license management tool is used to generate an encrypted license file;
2. The license file, together with the customer registration number is sent to the customers. Note: the customer registration number is sent only once and it is used to uniquely identify a customer;
3. The customers use the admin tool to send the registration data to the SystemWeaver server;
4. The SystemWeaver model server, reads the registration data and validates it;
5. SystemWeaver server writes the content to the SystemWeaver database.

6.3 Detailed design

6.3.1 License management tool

The standalone license management application is responsible for license generation. It will also be able to accommodate tracking of licenses customers, invoicing etc. The standalone application stores client and licenses' general information and will offer CRUD (**C**reate/**R**ead/**U**pdate/**D**elete) functionality for its entities.

The License management tool is structured into several layers. Figure 6.2 presents its three tier architecture. In this section are presented some of the advantages and motivation behind this approach. Each component of the architecture, starting from the bottom to the top level is presented in this section.

Each layer of the architecture provides services to the layer above by adding more functionality and exposing more specialized functionality of the layer below. Each level communicates only with the level situated immediately below and each level exposes its services to the layer above through a SAP (Service Access Point).

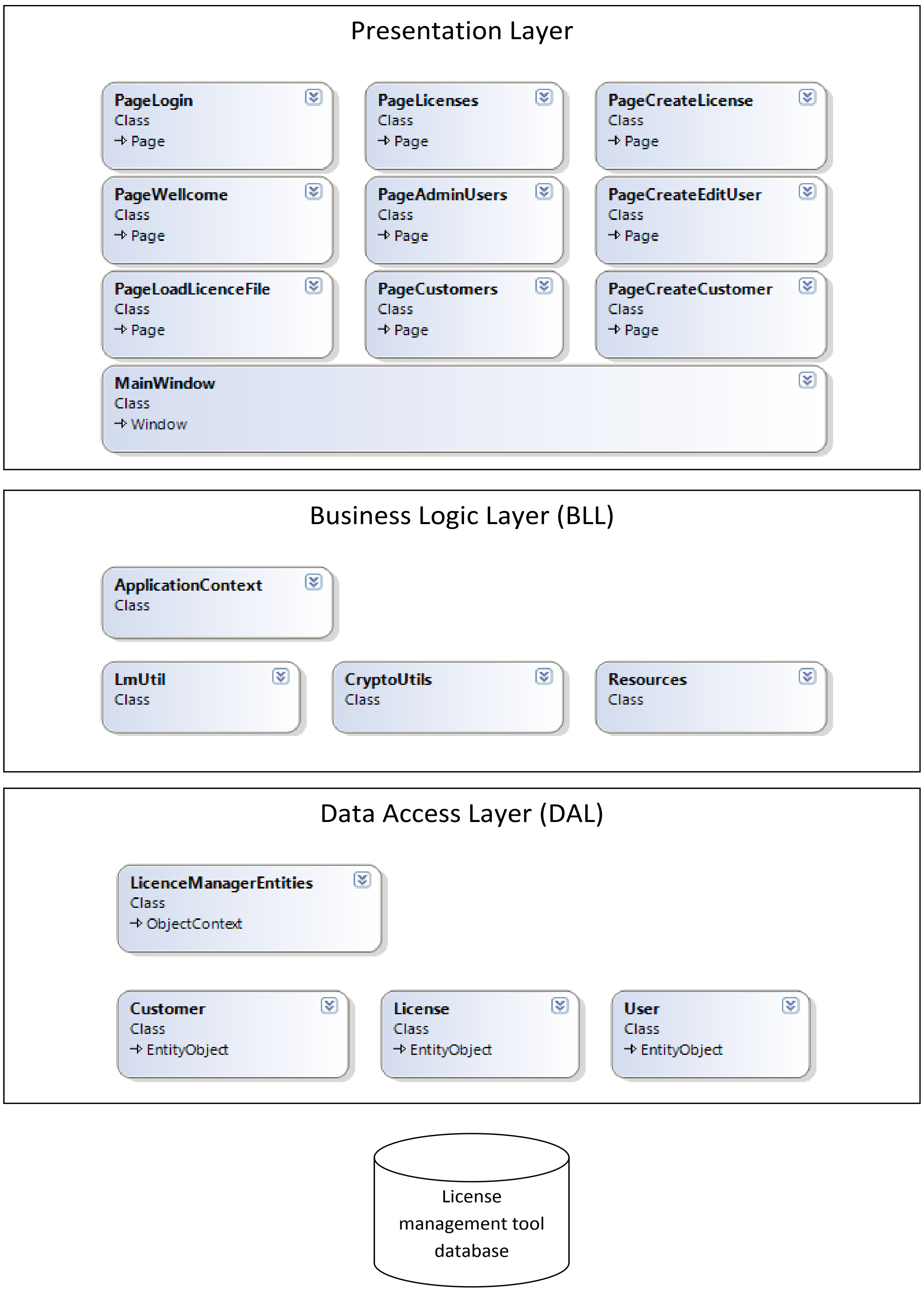


Figure 6.2 License management tool architecture

The **License management tool database** is structured in three tables: “Customers”, “Users” and “Licenses”. In Figure 6.3 is presented a diagram of the database that shows the tables’ columns and the relation between them. The “Customers” table contains different information related to customers. “Users” is a table that contains the users of the license management tool, while“License”has as purpose the storage of the soled licenses. The

“UserId” and the “CustomerId” from the Licenses table are foreign keys pointing to the user that created the license, respectively, to the intended customer.

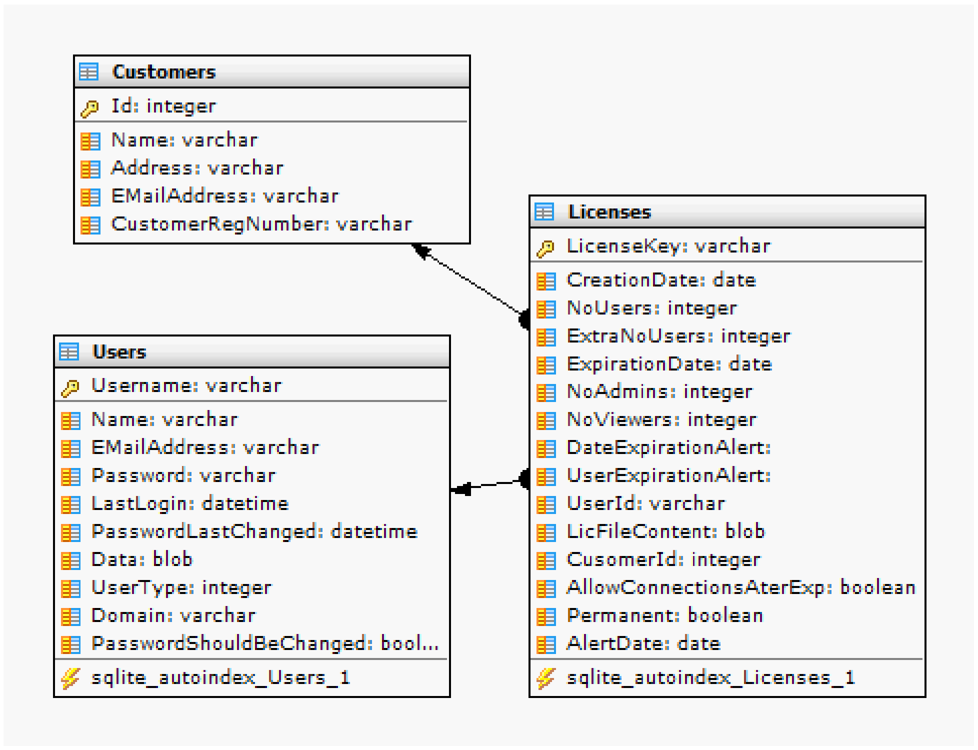


Figure 6.3 License management tool database

The **Data Access Layer** contains all the code needed to access the database. It simplifies the design since one does not have to think at database access while designing the Business Logic Layer. It contains **LicenseManagerEntities** which serves as a SAP to the business logic layer.LicenseManagerEntities offers to the business layer functions such as retrieving all the information from the database and also commit (make the changes persistent) functionality in case there are changes.

The rest of this layer’s components are data entities. Entities are richly structured records with a key. The entities are grouped in entity sets. In Figure 6.2, in the data access layer can be seen three entities like: Customer that represents a record in the “Customers” table, License that represents a record in the “Licenses” table and User that represents a record in the “Users” table.

The **Business Logic Layer**contains most of the code for this application. It uses the functionality provided by the Data Access Layer and the utility classes **LmUtils** and **CryptoUtils** exposing more specialized services to the layer above. The business layer exposes its functionality to the layer above through **ApplicationContext** which is a singleton. This provides the presentation layer with all the needed functionality. Presentation layer only has to delegate requests to the ApplicationContext. Some of functionality exposed by the AplicationContext is as following:

- Login and Logout;

- Generate license key and customer registration number;
- Retrieve all customers, users and licenses;
- Get licenses by customers;
- Get license content byte array.

The **Presentation Layer** consists of a frame (**MainWindow**) and several pages as it can be seen in the presentation layer shown in Figure 6.2. The **MainWindow** contains a frame that serves as container for the pages. The pages and their usage scenarios are presented in more detail in the implementation section together with the used technology.

The three tier architecture comes with several advantages. Using this approach it is easier to replace one layer of the architecture with a new one without affecting the others.Another advantage of this architecture is that other layers can be inserted without affecting the existing ones. For example a WCF Service Hosting Layer can be inserted between the presentation and the Business Logic Layer. Presentation layer can communicate with the lower levels (BLL) through WCF over the Internet. It will only invoke services exposed by the WCF Service Hosting Layer that further delegates the requests to the Business Logic Layer (BLL). For more details refer to [11].

6.4 SystemWeaver prototype

As it can be seen in the **Figure 6.1,** it is required to extend the functionality of the SystemWeaver server and the SystemWeaver admin tool.Therefore, it is neededto modify only two of the applications, fact which proves advantageous also when it comes to decreasing the costs. However, in this project it is built only a prototype that simulates SystemWeaver server for proof of concept. The prototype is built from scratch and it contains a server and two client applications.

The reasons why we chose to implement a prototype instead of integrating directly SystemWeaver were the following:

- Given the fact that the company sends constantly new release updates to its customers, an integration would have been difficult to achieve;
- It is intended to introduce the license manager only after this has been fully integrated and tested. Since Systemite’s purpose is to offer high availability to its customers, it is beyond the scope to add, at this stage, a daemon that might be triggered in unknown conditions thus generating errors and preventing the users to accomplish their work;
- An alternative possibility to cope with the frequent updates that the company sends to its customers would have been to create a special branch aimed only for license manager. Again, even in this case, this would have increased significantly the complexity: several builds would have been created thus increasing the possibility for mishandling and bad deliveries to the customer;

- Also, using a prototype instead of the real application, the focus would be more on testing the functionality of the license manager itself rather that dealing with integration issues.

From an architectural point of view, the prototype uses the same principles as SystemWeaver, namely: it is based on a client – server architecture, supporting different types of clients intended to different kinds of users. **Figure 6.4** presents the SystemWeaver main prototype applications like the normal client, the administrative tools and the server but also the interactions between the client and the server.

The **Server Prototype** offers basic functionality like authentication, adding users and license file registration. **swDefs** is a unit which contains functions used for the communication protocol like generating messages and information retrieval from the messages.

CryptoUtils and **CryptoUtils256** are two help units that tare exposing the same functions used for encryption and decryption. The only difference is that CryptoUtils provides support for 128 bit key while the other provides supports for 256 bit key support.

The **TUserEntity** is abstractization of a user. **TLicenseWrapper** represents a wrapper over the license content that knows how to interpret the file and provides fast access to the licensing information.

The **ServerForm** contains the actual user interface of the server application. It contains also the TCP server and all of the requests handlers. It stores the users and the licenses in the database. We choose to store the database information in the database rather than in the Windows registry. Storing the licensing information in the Windows registry violates product development’s need according to which the clients should not contact Systemite if they have to move the server to another machine.

Storing the licensing information in the database presents both advantages and disadvantages. First of all, it is easy to move the server to another machine since the licensing information is stored in the database and it will be moved at the same time with the server. In the case a separate database server is used, they have to move only the SystemWeaver server. Another advantage is that Systemite is able to easily send new releases to its customers.

One of the disadvantages is that one can replicate a database and give a copy to someone else. However we accept this problem since it is part of some of Systemite’s customers’ workflow. They are developing the metamodel and afterwards they send a copy of the database to quality insurance department for testing.

In the current prototype the database contains only authentication information and licensing information.

The **Admin Prototype** simulates the **swAdmin** tool. The prototype provides functionality like authentication, addition of users and registering the license file.

The **Client Prototype** simulates the **swExplorer** tool which is the client that is mostly used by the normal users. In the scope of the license manager the only functionality that we need from this client is the authentication.

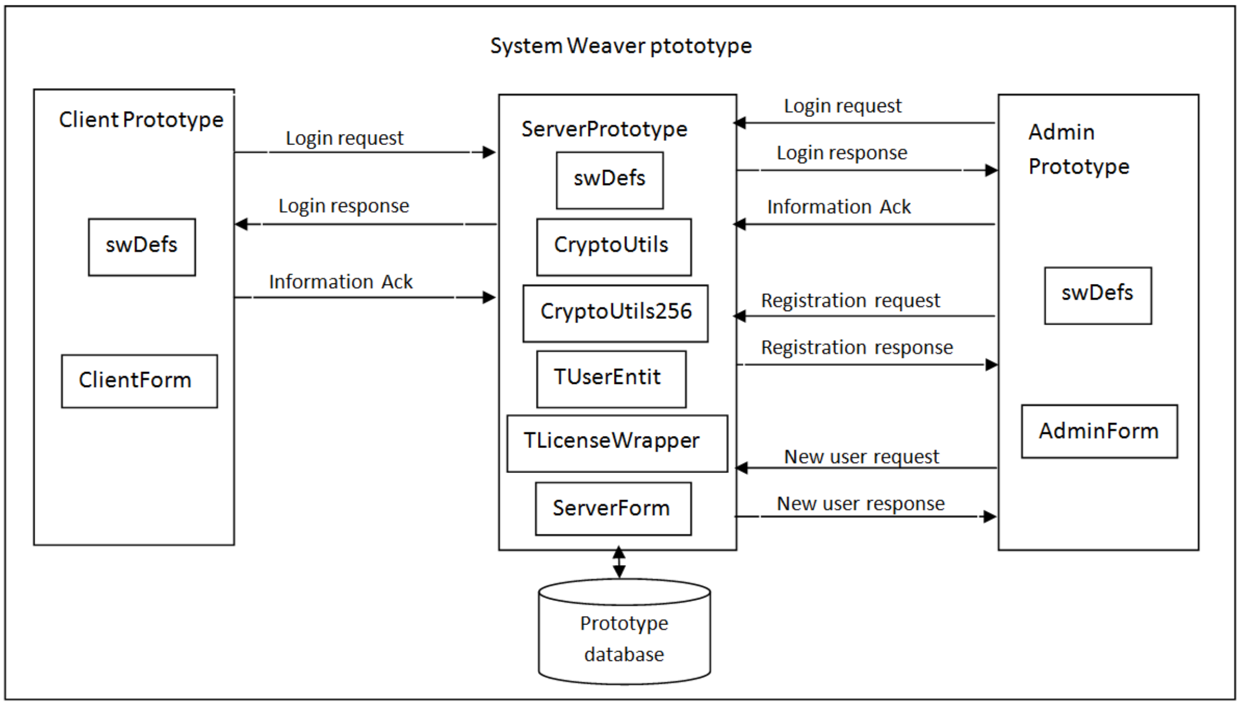


Figure 6.4 SystemWeaver prototype

Since the prototype is build from scratch there was a need for a **communication protocol** between the clients and the server.

The prototype uses a XML base communication protocol. Each message is formatted as XML. Using this has in general several advantages like:

- In general the clients are completely agnostic about the server supporting different types of clients.
- In our particular case it is a flexible way of exchanging data, information can be easily added and removed from the messages.

For **authentication** the clients send a login request containing the user id and the password. When receiving an authentication request the server looks at the resources available resources. The server verifies the authentication and the available resource. In the case of a successful login it can also add an information messages such as the date at which the application will expire. If the server requires acknowledge for the info messages, an ACK flag is set to true. The client application displays the message and if the ACK flag is set, it sends an acknowledgment that it has displayed the message. When the server receives this acknowledge, it computes the next alert date for the current users and saves it in the database.

For **registration** the admin tool will sent a registration request. The request contains license file content and also the customer registration number. The server analysis the request and registers it in the database if it is valid and sends back a registration response.

For creating new users the admin tools sends a new user request. The server checks that there are still resources left and sends an appropriate response.

Chapter 7

Implementation

In this chapter we present the technologies involved in the implementation step. We will also present the motivation behind the employed technologies as well as the problems encountered throughout this phase.

7.1 Technology choices and motivation

According to the conceptual architecture, the License Manager is formed of a standalone application and a client - server prototype that simulates the SystemWeaver server. The stand alone application is implemented in C# over .NET Framework 4.0.

.NET Framework is an integral Windows component and its purpose is to offer a framework for building desktop and web applications. It provides developers with an object oriented programming environment while minimizing the deployment effort and versioning conflicts.

It is a cluster of several technologies such as:

- .Net languages like C#, F#, Visual Basic and C++;
- .Net framework class library contains prebuilt functionality that the programmers can use into their applications;
- ASP.NET is the engine that hosts the web applications;
- Common language routine (CLR) is the engine that executes all .NET programs;
- Visual Studio is an optional development tool.

C# is a modern, powerful component oriented language containing strong typing, declarative, imperative, and functional paradigms.

SystemWeaver is implemented using Delphi. The language behind is Object Pascal, that is a variant of Borland Pascal which adds object oriented capabilities. The syntax is a bit different from C/C++ but it is well-structured, minimizing the development time without altering the performance.

The prototype that simulates the SystemWeaver is implemented using Delphi XE2 in order to be consistent with SystemWeaver. We could not choose .NET Framework for the prototype because when some of the concepts will be moved to the real SystemWeaver, problems regarding the incompatibilities between technologies may arise. Using Delphi, this migration will be easier since most of the functionality is already implemented.

7.2 License management tool

7.2.1 Database and data access layer

SystemWeaver uses SQLite database, therefore it was a requirement to use SQLite for the stand-alone application that was developed for this project. SQLite is a self-contained, server less, zero configuration transactional database engine. It is one of the most deployed database engines in the world. Connecting to a SQLite database proved not to be easy since ADO .NET does not have a provider for this database engine.

Hence, for the implementation, a third party provider was needed. The open source provider that matched our needs and that was further used was System.Data.SQLite. System.Data.SQLite consists of a complete SQLite database engine and a complete ADO.NET 2.0/3.5 provider. It can be used as well as a standalone database engine since it is a drop-in replacement of the original one. The ADO.NET provider supports all of the most recent changes added to the ADO.NET framework. It supports nearly all of the Entity Framework functionality that the SQL Server supports.

ADO.NET Entity Framework enables DAL to grant its functionality to the layer above. Moreover it helps the software developer to create “data access applications by programming against a conceptual application model. Nevertheless, it decreases the amount of code that has to be maintained for data oriented applications. Another advantage is that Language-integrated query is supported providing compile-time syntax validation for queries against a conceptual model” [12].

The class diagram of the data access layer and the relation between entities can be seen in Figure 7.1. **Customer entity** represents a row (record) from the customers table. As one can see, most of the Customer entity properties correspond to a column of the Customers tables. It contains information related to the Systemite’s customers. The stored information may be changed and one can add more information without modifying the business logic or the presentation layer.

The **User entity** contains license manager end users’ related information. It contains the necessary information for authentication and identifications but also for tracking users and their actions. The Password property does not contain the password in plain text but it contains the hashed password.

The **License entity** contains most of the properties that were detailed in the Analysis and design and some more. It contains as well the username of the user that has issued the license as well as the foreign key of the target customer. The additional information that is not contained in the License entity is taken from the customer due the relation between them.

The above layer accesses the data through **LicenseManagerEntities**. This contains three properties like: Customer, Licenses and Users. The properties are sets that contain all entities which covers all the data from the database. It contains as well methods for adding customers, users and licenses. The SaveChanges method will make all the changes persistent in the database.

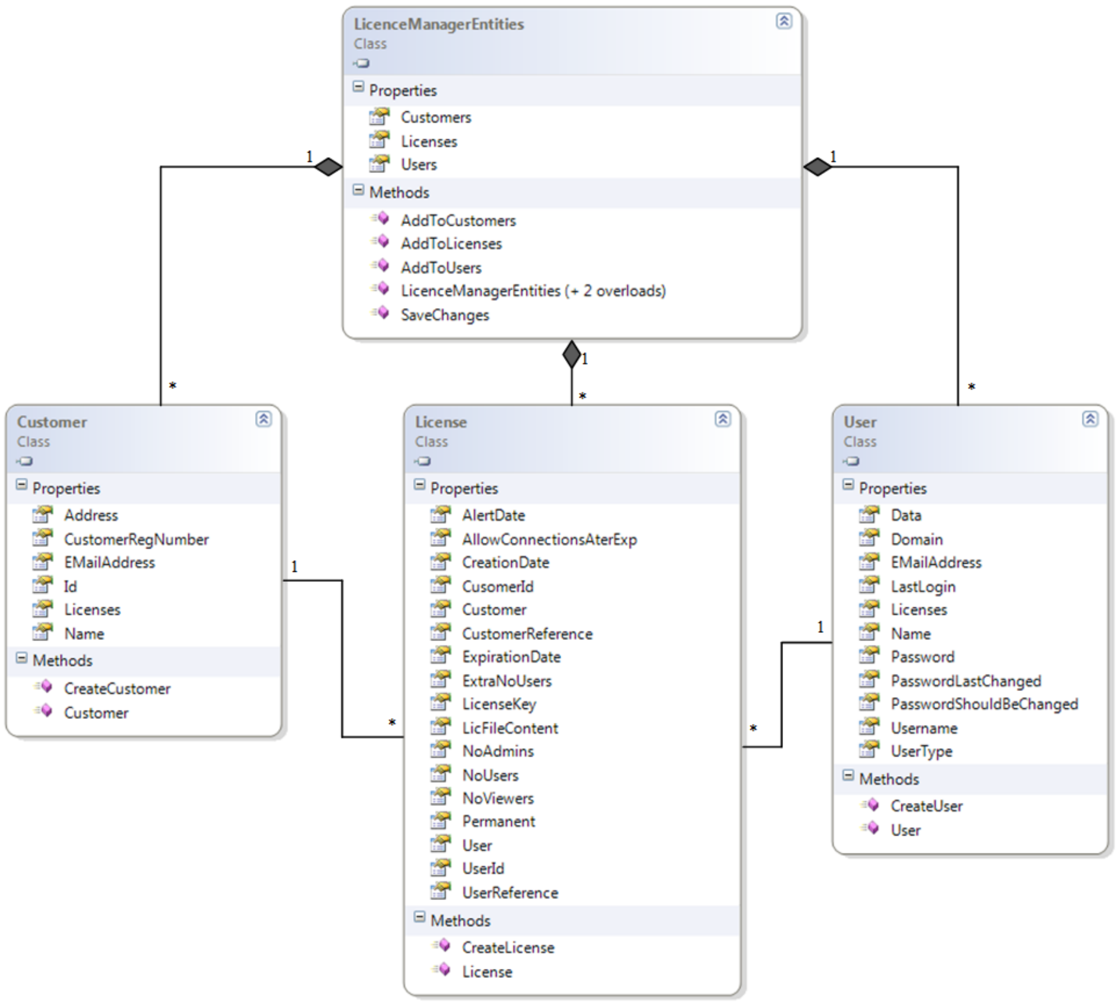


Figure 7.1 DAL class diagram

7.2.2 Business Logic Layer

The Business Logic Layer is the place where most of the code for this application is residing. It uses the functionality provided by the Data Access Layer exposing more specialized services to the layer above using LINQ to Entities and other utility classes like CryptoUtility and LmUtil. LINQ to Entities is referencing the entities model provided by the DAL.

Whenever a LINQ query is executed, this will be translated by the Entity Framework to the Business Model (entities) that represents the conceptual entity model. Further on, the entity aspects are mapped to the database tier and a SQL query that is to be executed towards the database is generated.

The business logic class diagram is presented in Figure 7.2

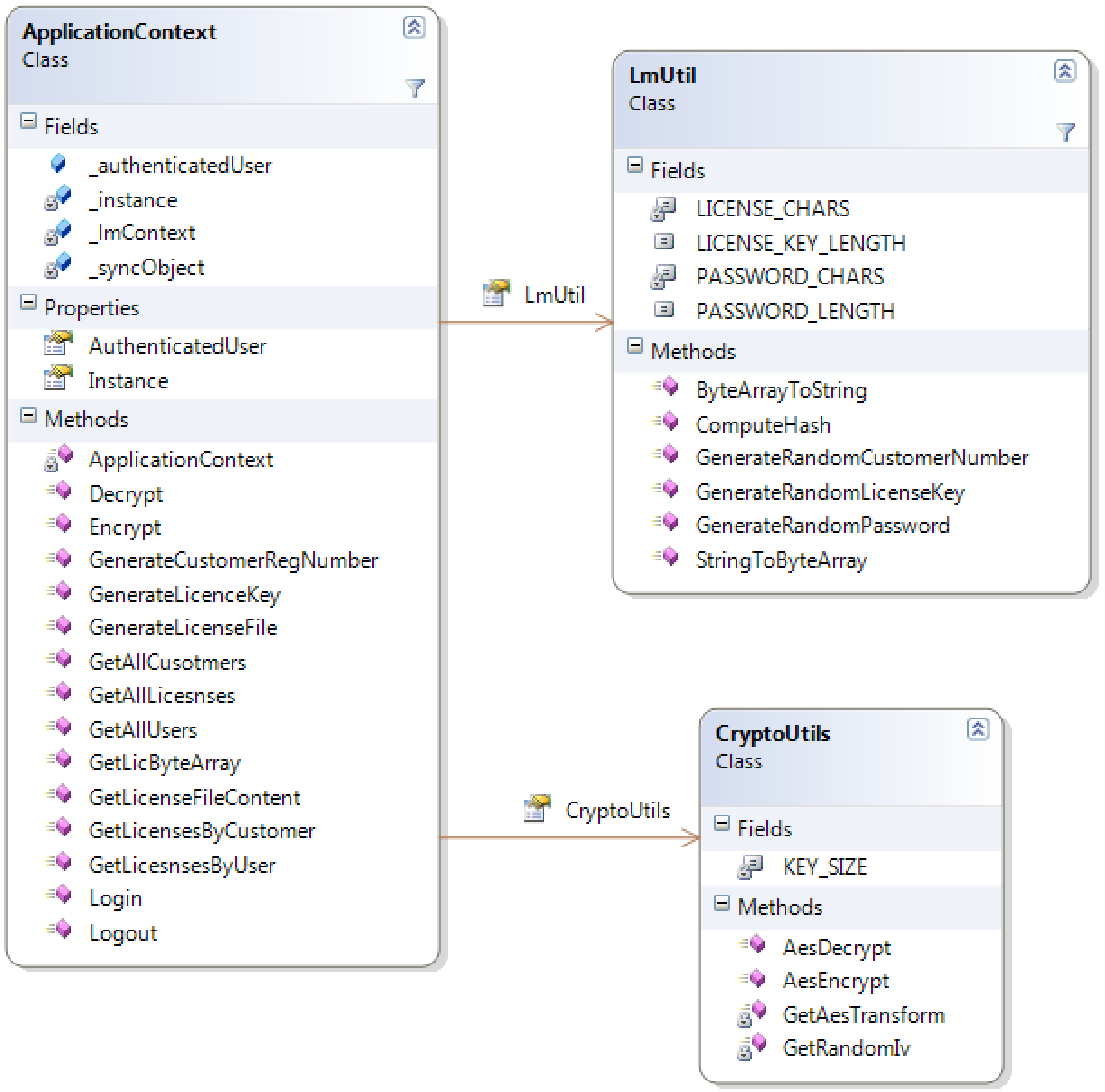


Figure 7.2 BLL class diagram

7.2.2.1 ApplicationContext class

The business layer exposes its functionality to the layer above through **ApplicationContext**, which is a singleton. This provides the presentation layer with all the needed functionality. Presentation layer only has to delegate requests to the **ApplicationContext**.

ApplicationContext is a multithread singleton. This is needed in a multithread environment due to the fact that the private constructor executes initialization code like the instantiation of **LicenseManagerEntities**. Therefore, we do not want to open several points of access to the DAL. We need to ensure that only one instance of it can exist in the system in the presence of multiple threads. In the current implementation this would not be the case, but when moving to a more distributed environment it might cause problems. A technique called “Double-Check locking” [13] [14] is used to avoid several threads creating more than one instance of the ApplicationContext.

Encrypt and Decrypt are the two functions used for encryption, respectively decryption. The encryption and decryption is not implemented in this class but the ApplicationContext delegates the call to the CryptoUtils utility class. The methods GenerateLicesingKey, GenerateCustomerRegistrationNumber are offering the functionality by delegating the call to the LmUtil. The rest of the methods are using LINQ.

7.2.2.2CryotoUtilsclass

For encryption and decryption we use the **AesCryptoProvider** and **ICryptoTransform**. The **GetAesTransform** takes as arguments a string and a Boolean. The string contains a password that is used for generating the encryption key. The password is hashed using SHA256 and takes the first KEY_SIZE bytes as encryption key. KEY_SIZE is a constant that defines the size of the key. In case it is wanted to change the key size, this can be done easily by modifying this constant.

The used encryption mode is CBC; therefore there is a need of an initialization vector. Since it is must be a random number, we used a used a function **GetRandomIv**. **GetAesTransform** returns an **ICryptoTransform** that defines basic operations for cryptographic transformation. Based on the decryptor flag it returns a decryptor or an encryptor. **AesCryptoServiceProvider** performs symmetric encryption and decryption containing an implementation of Advance Encryption Standard.

AesEncrypt takes as arguments a plaintext and a password. It calls the **GetAesTransform** in order to get an encryption cryptographic transform and returns a string that contains the ciphertext.

AesDecrypt is the inverse function of AesEncrypt. It takes as arguments a ciphertext and the passwords. It calls the **GetAesTransform** in order to get a decryptor. It decrypts the cipher text and it returns a string containing the plaintext. All the three functions are static and are located in a utility class.

7.2.2.3LmUtil class

LmUtil class - most of the utility functions implemented in this class are used for generating random string like: passwords for new user, license keys, customer registration numbers. It uses **Random** class which is a random number generator. This class is provided by the .NET Framework.

7.2.3 Presentation layer

The presentation layer is implemented using **WPF** (Windows Presentation Foundation). WPF is providing powerful tools that can be used by designers and developers to create friendly interfaces, interfaces that can be of varied content (documents, multimedia, 2D and 3D graphical objects, etc.) , without being limited to buttons and simple lists.

One of its main advantages offered by WPF consists in that the interface is kept in a separate file type, **XAML** (eXtensible Application Markup Language). In this way an interface designer, using a dedicated environment for interface design (WPF Designer that is included in Microsoft Visual Studio 2010 or Expression Blend), can create interfaces that are saved as XAML files. These files will be used directly and the programmer will write only the interaction code for every action executed by the user. Using XAML has another advantage: the application can be easily migrated to a web one.

As describes in [15], the WPF navigation system provides the application's user interface with a browser style design. For example, WPF Frame component serves as a container for WPF Page. Using the navigation system provided by the former, one can navigate through the available pages by using hyperlinks. Moreover, WPF keeps track of the viewed pages and gives the possibility of back and forward navigation.

The current application's user interface consists of a frame which serves as a container for nine pages:

- PageLogin
- PageWelcome
- PageLicesnses
- PageCustomers
- PageUsers
- PageCreateUser
- PageCreateLicense
- PageCreateCustomer
- PageLicenseFile

When starting the application the login page Figure 7.3 is displayed. After entering the correct login information the user reaches the welcome page that can be viewed in the Figure 7.4.

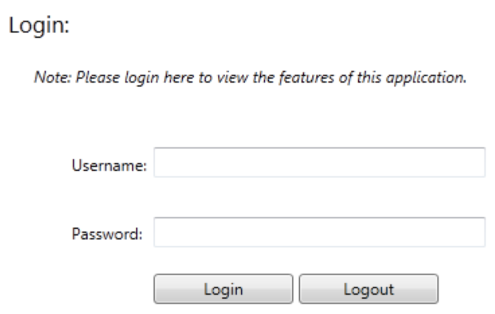

Login:
Note: Please login here to view the features of this application.
Username:
Password:
Login
Logout

Figure 7.3 Authentication page

In the welcome page there are several hyperlinks to other pages like: *Users page*, *Customers page*, *Licenses page* and a special page where the users can load a license file to read its content.

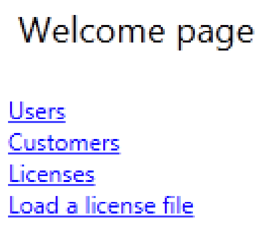
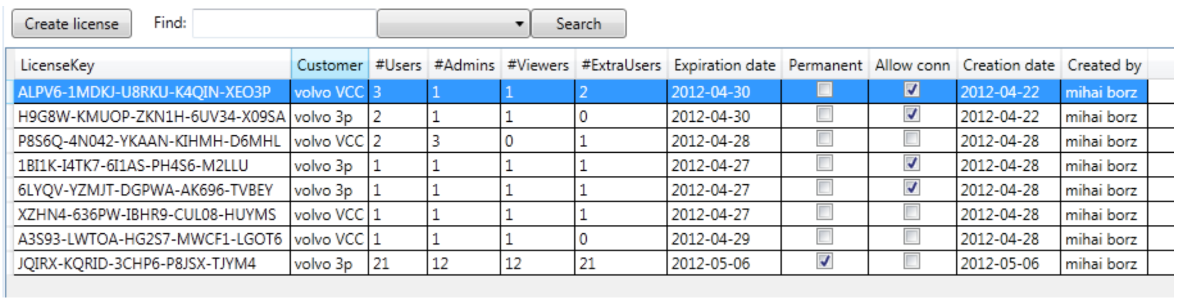


Figure 7.4 Welcome page

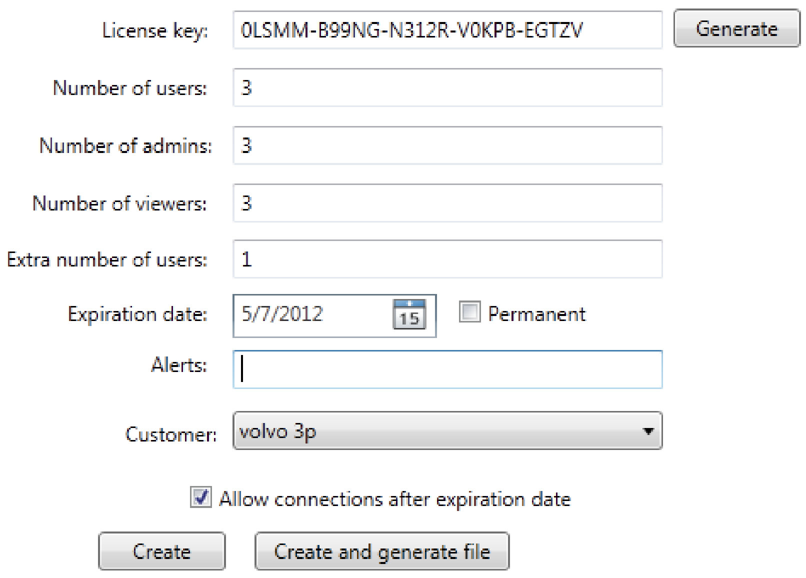
Only one of the above pages is presented since the other pages are quite similar and the usage workflow is almost the same. If a user clicks on the License page, the page shown in Figure 7.5 is displayed. This page contains a grid that displays the license and the related information. It also has a toolbar where users can filter by a text. They can filter by column or they can search through all columns if they do not pick a particular one. It also provides the users with sorting capabilities.



LicenseKey	Customer	#Users	#Admins	#Viewers	#ExtraUsers	Expiration date	Permanent	Allow conn	Creation date	Created by
ALPV6-1MDKJ-U8RKU-K4QIN-XEO3P	volvo VCC	3	1	1	2	2012-04-30	☐	☑	2012-04-22	mihai borz
H9G8W-KMUOP-ZKN1H-6UV34-X09SA	volvo 3p	2	1	1	0	2012-04-30	☐	☑	2012-04-22	mihai borz
P8S6Q-4N042-YKAAN-KIHMH-D6MHL	volvo VCC	2	3	0	1	2012-04-28	☐	☐	2012-04-28	mihai borz
1BI1K-I4TK7-6I1AS-PH4S6-M2LLU	volvo 3p	1	1	1	1	2012-04-27	☐	☑	2012-04-28	mihai borz
6LYQV-YZMJT-DGPWA-AK696-TVBEY	volvo 3p	1	1	1	1	2012-04-27	☐	☑	2012-04-28	mihai borz
XZHN4-636PW-IBHR9-CUL08-HUYMS	volvo VCC	1	1	1	1	2012-04-27	☐	☐	2012-04-28	mihai borz
A3S93-LWTOA-HG2S7-MWCF1-LGOT6	volvo VCC	1	1	1	0	2012-04-29	☐	☐	2012-04-28	mihai borz
JQIRX-KQRID-3CHP6-P8JSX-TJYM4	volvo 3p	21	12	12	21	2012-05-06	☑	☐	2012-05-06	mihai borz

Figure 7.5 Customers page

To create a new license the user has to press the create license button. A new page is displayed, see Figure 7.6, that contains a form for the creation of a new license.



License key:	0LSMM-B99NG-N312R-V0KPB-EGTZV
Number of users:	3
Number of admins:	3
Number of viewers:	3
Extra number of users:	1
Expiration date:	5/7/2012
Alerts:	
Customer:	volvo 3p

Figure 7.6 Create license page

The user can always drop the operation and can return to the previous page using the navigation bar presented in Figure 7.7. The users can either use the arrows or use the drop down menu to select one of the previous pages.

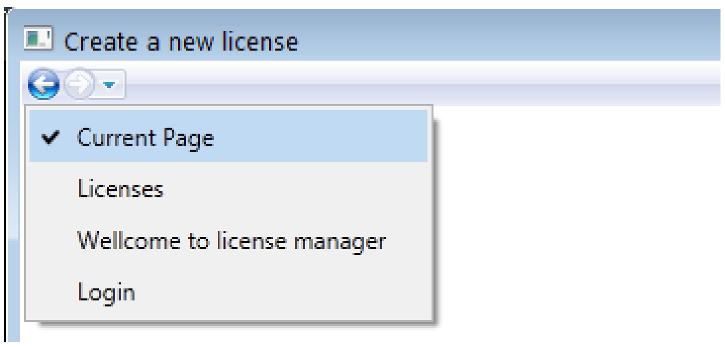


Figure 7.7 Navigation bar

7.3 SystemWeaver prototype

7.3.1 Database and data access

To access a SQLite database in Delphi requires too much work. The scope is to build a prototype as a proof of concept not to connect to a SQLite database. The connection to a SQLite database is already done for the SystemWeaver server. For those reasons we choose to use for the prototype to use a Microsoft Access database and we use SQL query to access it.

The SystemWeaver prototype database structure is simple and contains information only needed for licensing and for authentication. Therefore it contains two tables: **Users** and **Usages**. The Users tablecontains information related to the users like: Username, Password, Type, Name, InformDate, Email etc. The Type refers to the actual type of the user like root, admin viewer and normal user. The Inform data contains the dateat which the user must be informed about the license expiration.

The "Usages" table contains licensing information. It contains a blob where the encrypted content of the license file relies. Besides the blob it also contains other column with the resources that are available and configured in the license file. Apart from the license information it contains also a registration date. The SystemWeaver server is always using the last registered license.

7.3.2 Cryptographic algorithms

The cryptographic library caused a lot of problems while implementing the SystemWeaver prototype. A first attempt was to use **TurboPowerLockBox 3**. According to [16], TurboPowerLockBox 3 is a Delphi Cryptographic Library that offers the main encryption/decryption related functionality such as: public key encryption, symmetric encryption and hashing functions. However, after spending some time trying to make it work we gave up the idea of using it due to the poor documentation and incompatibilities between the technologies. Therefore we chose to use the Windows ADVAPI32.dll. It provides additional functionality to the Windows kernel. ADVAPI32 is an advance API that supports numerous functionality including security and registry calls.

7.3.3 License registration

The license registration can be done by an admin or by the root user. In the admin tool the user has to select the file and input the customer registration number. The registration date is send afterwards to the server prototype. On the server side, the server verifies that the customer registration number provided is the correct one. The server adds a new record in the Usages table. Afterwards, it saves the encrypted content in the blob filed. It also adds the resources of the license file in the other columns in plain text as well as the registration date.

The "Usage" table can contain several entries. It keeps as well the previous registered licenses. When the server retrieves license information, it sorts the records in the Usages table based on the registration date and the last registered license. This mechanism is also good in the case of permanent licenses when the customers need more users for a certain time period. Then they will receive the new license with the number of the users that they need for the requested duration. After expiration they have to remove the last license and return to the permanent one.

When retrieving the licensing information the server always takes the information from the encrypted field. Each time the server is accessing the licensing information it has to decrypt its content.

7.3.4User authentication

The authentication is a complex process. On each login, the server retrieves the license information from the Usages table. The authentication process flowchart is presented in the Figure 7.8. When the server receives the authentication request it retrieves the user from the database. If the authentication data is incorrect it stops and returns the appropriate messages. In the case the authentication data is correct, it checks if the user is root. If the user is root and the client is the admin tool it returns true. We need an account that is able to connect even after the expiration date or in the case the number of users exceeds the agreed one, to be able to remove or deactivate users or register a new license file.

In the case of another user we cheek the expiration date. If the license is not permanent and the license has expired it verifies the “Allow connections after expiration” flag. In the case it is set to true the user is still able to login but is downgraded to viewer. Otherwise the authentication will fail.If the license is permanent or the data is still valid it verifies the number of users. If the number of users exceeds the agreed one, the authentication process will fail and an appropriate message will be generated. This check is done to avoid addition of users directly in the database.

After verifying the users it checks the InformDate field in the user’s record. In the InformDate is less than the current date the server will generate the inform message and will set the ackRequired flag. The authentication process succeeds.

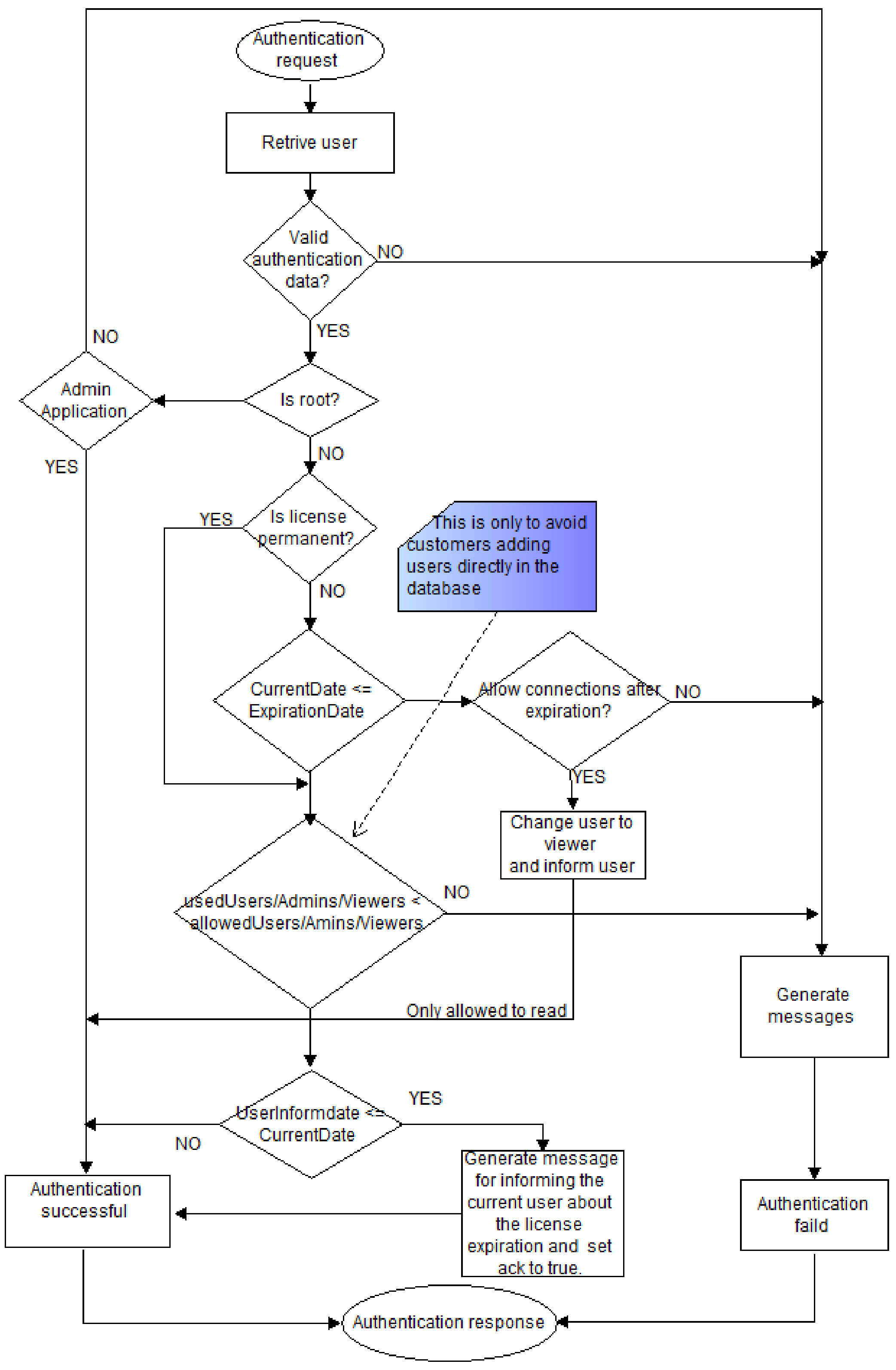

Authentication request
Retrive user
Valid authentication data?
NO
YES
Is root?
NO
Admin Application
NO
YES
Is license permanent?
YES
NO
This is only to avoid customers adding users directly in the database
CurrentDate <= ExpirationDate
Allow connections after expiration?
NO
YES
Change user to viewer and inform user
usedUsers/Admins/Viewers < allowedUsers/Amins/Viewers
NO
Only allowed to read
Generate messages
UserInformdate <= CurrentDate
YES
NO
Generate message for informing the current user about the license expiration and set ack to true.
Authentication successful
Authentication faild
Authentication response

Figure 7.8 Authentication flowchart

If the SystemWeaver client receives a login response with the ackRequired flag on, it displays the information messages (when the license expires) and sends an acknowledge message containing the username back to the server.

When the server receives the ACK message the algorithm presented in Figure 7.9 is executed. There will be three expiration date alerts. The first expiration date alert will be at start alert date. The second alert will be at the half of the interval between date alert and expirations date. The final alert will be at three quarters of the interval between alert date and expiration date. There are no alerts regarding the number of users to inform normal users. In the admin tool there is a panel that will inform the administrator about the number of resources left (time and users).

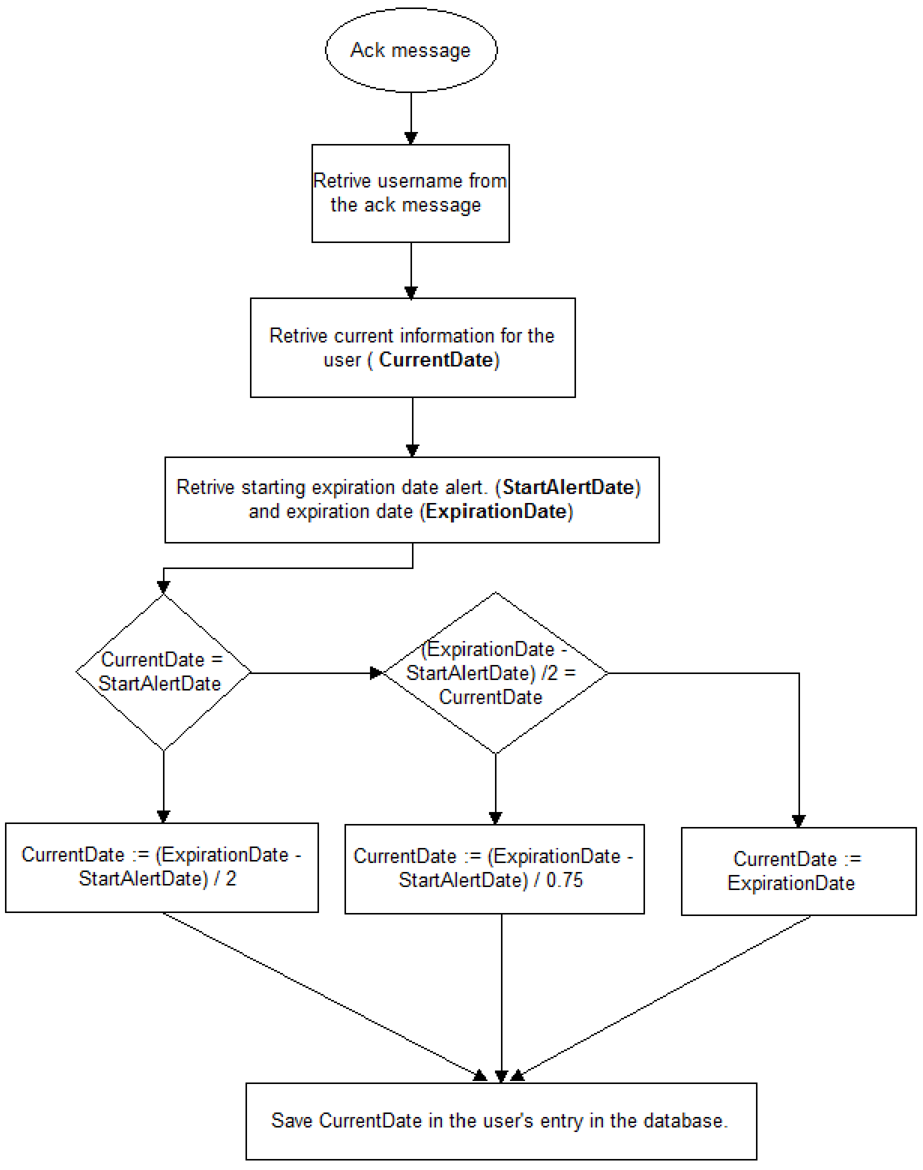

Ack message
Retrive username from the ack message
Retrive current information for the user (CurrentDate)
Retrive starting expiration date alert. (StartAlertDate) and expiration date (ExpirationDate)
CurrentDate = StartAlertDate
(ExpirationDate - StartAlertDate) /2 = CurrentDate
CurrentDate := (ExpirationDate - StartAlertDate) / 2
CurrentDate := (ExpirationDate - StartAlertDate) / 0.75
CurrentDate := ExpirationDate
Save CurrentDate in the user's entry in the database.

Figure 7.9 Receiving ACK flowchart

Chapter 8

Testing and validation

This chapter presents the approach used to test the system in order to confirm itsfunctionality and, the system analysis based on the attacker model presented in the System analysis chapter. The costs that are introduced by adding the license manager mechanism are also mentioned.

The testing and validation of the current proposed system was performed in two stages. At a first stage the testing and the validation of the proposed solution were performed in a theoretical way. Afterwards, different usage scenarios were proposed based on the system specification (uses cases). We tried to run them conceptually, following the same steps the final user has to perform in order to complete the proposed scenarios. In this manner we analyzed whether the solution satisfies the license manager requirements. We also tried to analyze if the proposed solution is resistant to the defined attacker model (see chapter 5.3.5).

After the first step, the solution was validated but, this did neither exclude the existence of bugs, nor that is not feasible to be implemented. It is not eliminated the existence of other problems or security issues that might not have been easily seen from the very beginning. Nevertheless, it is important to prove that the proposed solution is implementable given the mixture of technologies involved (Delphi, C#).In this scope, a final application (for license generation) and a prototype that simulates SystemWeaver system were built for testing and validation. The reason behind implementing a prototype was to limit the testing scope. By using just a prototype having the needed functionality, the testing of the License Manager focuses only on its functionally and not on the functionality of the overall system. The scenarios from the previous step were used but this time they were run and tested on a real application.

8.1 Testing the performance

This section is aimed for analyzing how the license manager may affect the performances of the SystemWeaver. In other words, we are interested in identifying the exact overhead the license manager adds to the current solution. We intended not only to measure the overhead that was introduced by the license manager itself, but also to make a comparison between the overhead introduced by varying the key sizes. It should be mentioned that in the current case, a certain limitation in testing existed mainly because we tested on a prototype that has only the authentication and lacked all the other features of SystemWeaver. The proposed solution affects only the authentication, therefore we tested the time duration between the authentication request and the authentication response. Note that this test serves only for having a high overview of the penalty performances that may be introduced solely by the license manager mechanisms. In a regular usage scenario other delays may occur i.e., when the server is busy decrypting the licensing information required for authentication, other operations may suffer from inherent delays.

For testing the performance, both the server and the client prototype were run on the same machine. This will also ensure that delays due to network communications that might vary from one test to another and thus prevent us from making truthful observations, were avoided.

Firstly the tests were run having enabled only the authentication mechanism i.e. we verified only whether the authentication data is correct. The purpose was to measure the time that is consumed by the authentication itself. Secondly, tests were run while having the license manager enabled and using a 128 bit key for the encrypting/decrypting licensing information. Same test was also performed but using a 256 bit key. The purpose here was to motivate the selected key size while checking how an increase in security leads to a performance penalty.

The testing was performed on an i7-2600 @ 3.4 GHz 8MB cache computer. The results were summarized in the table below. As it can be seen, without the license manager the authentication takes on around 16[ms]. This number includes the amount of time required for validating the user’s credential after retrieving them from the database.

As expected, enabling the licensing manager with a 128 bit key causes certain latency and affects the overall performance: the authentication time is doubled. As in the previous method, part of the time duration comes from requesting and validating user’s credentials. The difference comes in from: retrieving the licensing information decrypting the content and checking the current resources against the ones stated in the licensing information (refer to the Implementation chapter, Figure 7.8).

Enabling the license manager with a 256 bit key causes even higher differences when it comes to performances (the time duration is tripled compared to the first case). Apart from the decryption, all the performed operations are identical to the ones in the previous scenario. As shown in the Table 7.1, a double key size adds a delay of up to 16[ms] compared to the previous case.

In conclusion, there is always a trade-off between security and performance: by adding extra security we decrease the performances and vice-versa. The decryption process proves to be an expensive process in terms of computational resources. Meanwhile, the content of the license file might increase in the future thus causing even higher performance penalties. As mentioned above, keeping the CPU busy with decryption will cause delays in the normal work flow of the application.

Key size	**Test1**	**Test 2**	**Test3**
Without verification	16[ms]	15[ms]	16[ms]
Verification 128 key	31[ms]	31[ms]	32[ms]
Verification 256 key	47[ms]	46[ms]	40[ms]

Table 8.1 Performance results

Therefore, we do not want to pay for the cost of a 256 bit key since this proves to be too high and on a loaded system, the delays might increase considerably. Also a 128 bit key provides good computational security against brute force attacks and easier ways of hacking the license manager - other than brute force on the key - exists.

8.2Security analysis

In the Analysis chapter we identified an attacker model comprising of a set of (i,p) (Intervention, Presence) pairs that can map over our system. This section describes what an attacker can do and what the behavior of the system is in case these attacks occur.

We identified two different types of presence:

- Local
- Remote

The **local** presence means that the attacker has direct access to the machine that is running SystemWeaver server. In this case the attacker has full access to the SystemWeaver database as well as to machine and to the operating system.

The **remote** presence means that the attacker does not have direct access to the machine that is running the SystemWeaver server. Note that using a remote desktop is still part of the local presence. The only way to harm the system in this presence is by using the SystemWeaver clients or another application that is talking to the SystemWeaver server or by sending directly other packets that might harm the system.

Enable extra resources in trial mode: when running in trial mode, an attacker should not be able to enable/use more resources than those allocated for trial mode. From a remote presence point of view, in order to add extra resources, admin tool must be used. Note that all the checks are done at the server’s side, admin tool having as purpose only the delegation of requests to the server. Therefore it would be impossible to add other resources using admin tool.

Another approach the attacker can use is that he could try to generate a license file to activate the resources. This proves also to be unfeasible since the secret key as well as the license file structure is needed in order to generate it. From a local presence point of view, in trial mode, there is no license information in the database thus the server will generate a temporary one. In this way the existing data cannot be altered in order to accept the addition of extra resources (users).

Addition of extra resources (in our case, user accounts) over those stipulated in the license agreement: although similar to the previous one, this intervention is possible only after license activation. For this, an attacker should firstly decrypt the content of the license file. The license file is encrypted using AES128, hence an attack, either coming from a local or from a remote location, that has as purpose the decryption of the file is not feasible (AES128 is secure).

Another approach is to alter the license content by flipping some bits in order to increase the number of resources. However, this is impossible not only because the amount and structure of information contained in the license file will differ from one file to another (more specifically, the license file is structured as an XML therefore the location within the file of the number of users will not be the same in separate files) but also due to the usage of CBC mode (the modification of one block will cause subsequent changes in the following blocks).

An attack that might be possible for both of the above interventions is to add resources directly in the database. This is only in the case where the attacker has access to the database (local presence). For example, an attacker may perform the following steps:

- Create a new entry in the Users table of the SystemWeaver server database;
- Generate/Create a new unique ID for the user;
- Choose a password and compute the hash using the ID as salt;
- Fill in the remaining data (username, email etc.);

As we presented in the implementation section we do not check the available resources only when adding new users but also during each authentication. Therefore, if the number of users is larger than the number stipulated in the license agreement, the server refuses to accept new connections. New connection requests will be refused until the number of users is less or equal to the agreed one.

*Affect time's flow on the server's side (make the time run backwards at server's side): this intervention affects the expiration date. More specifically, an attacker can postpone to an indefinite date the license's expiration by simply making the time on server's side run backwards.*This can be done only by having access to the machine that is running the SystemWeaver server (either remote or local). It is easy to realize, one has only to set the date in the past before reaching the expiration date. The current solution does not cover this intervention but this will be implemented in a next version. This intervention was not considered in the current implementation since it is not a good approach for Systemite's customers to run the time backwards. More specifically, it has an impact on the issue management and it will prove difficult to follow the progress of projects. Also, considering time registrations - no one will have access to the real time when an item or an issues was created.

Eavesdropping – this intervention is more related to SystemWeaver's security. However, it can also be regarded as a license manager problem since an attacker can eavesdrop the license registration messages and reuse their content. Since all Systemite's customers have confidential data in the SystemWeaver server, this uses TLS (Transport Layer Security) to ensure a secure communication between the SystemWeaver server and the client.

Chapter 9

Conclusions and further development

This section details the conclusions drawn after analyzing, designing, implementing and testing the SystemWeaver license manager. We will begin by making a brief overview of the topic and the steps followed in the implementation process. Also we will focus on the outcome of this thesis work showing the advantages and disadvantages of the given solution and what has been achieved compared to the initial goals. Further on, we will remind some of the encountered issues and, nevertheless, discuss the suitability of the license manager prototype for a real implementation. At the end of the chapter we give some suggestions for further development.

9.1 Conclusions and results

The aim of the current thesis was to investigate and implement a solution for preventing software piracy and managing software licenses, that meets Systemite's requirements. Overall, the scope was to have a flexible design, easy to implement and maintain and which, on the long run, does not add extra overload to the initial application or increases the costs.

Several steps were followed in order to fulfill the ahead mentioned goal. The first step consisted in gathering and analyzing the company's needs in terms of a license manager application and investigating the company's and its customers' ways of working. The outcome of this initial step was the selection of a single solution that maps best to the actual resources and to the current process. Several scenarios were created and ran against the proposed solution, verifying whether they meet our expectations in terms of security and company's needs. The following step was to design and implement the chosen solution. The aim was to better understand the behavior, analyze and test the outcome in real-life use cases.

Throughout the analysis and design stages we tried to find out the alternative that has the least impact on the already existing applications and current way of working. The chosen solution consisted in using a license file (refer to Chapter 6, System Design). The advantages this approach has over others are its increased flexibility and ease of modifying the product activation in the future. Meanwhile, the license file can be used as well for disabling and enabling different product features that are customizations for particular clients. Also the license file is used for configuring how the license manager should behave in different situations, e.g. once the time expired, can be configured in the license file to accept connections but the normal users will be treated as simple viewers. A license file validation procedure is preferred also because it is not dependent on an active Internet connection. Security wise, this approach has the disadvantage that a license file may be used to register multiple copies of the database. Anyhow, this may be difficult to avoid without using a connection to a registration server.

Throughout the project, the main challenges were to define the tradeoffs between security and availability, based on the stockholder's needs and to choose the appropriate software validation procedure. Another encountered issue was to mediate between different requirements coming from separate departments, e.g. sales department requesting high availability (access to the resources) for the 'trusted' customers, versus keeping the license manager active at all time.

Technology incompatibility was another major issue encountered during the development phase. Delphi has no support for providing encryption mechanisms/libraries. The third party library that was tried out proved to be incompatible with the .NET solution and had as well other limitations. Instead, for ensuring encryption capabilities, we chose to use the Windows ADVAPI32.dll library that provides additional functionality to the Windows kernel.

The outcome of this project is a complete stand-alone application, "License Manager Tool". This application is the final one and it implies no major changes in the near future. Besides the license manager tool we developed another set of applications that serve as a prototype and their scope is to simulate the current SystemWeaver platform and to prove the selected solution validity. Apart from these aims, building the prototype eased the testing and verification procedure by limiting the testing scope only to the license manager functionality.

9.2 Further development

As future developments, we intend, on short term, to continue the verification tests and migrate the concept and the implementation to the real SystemWeaver server. Also, as mentioned in the Testing and Validation chapter, the current solution is not resistant against running the time backwards at the server machine.

On the long term, we plan to integrate the license management tool with the financial system. Given the issues mentioned in the previous paragraphs, a long term solution will imply avoiding the registration using the same license file for several times. Hence, the clients will still be able to replicate the database and use it for both testing and development but they should not be any longer able of activating both copies.

Systemite sells licenses also through resellers. In the future it may prove to be useful to have a centralized database for tracking all the sold licenses irrespective of who sold them to the customers. The next step would be to change the current license manager tool in order to be able to generate license files via Internet. Nevertheless, the license management tool should support, in the nearest future, the possibility of buying license files through the Internet and online payment processing.

Bibliography

[1] http://systemite.se/content/products-services/systemweaver-concept

[2] http://en.wikipedia.org/wiki/File:Ecb_encryption.png

[3] http://en.wikipedia.org/wiki/File:Cbc_encryption.png

[4] http://en.wikipedia.org/wiki/File:Tux_ecb.jpg
http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

[5] http://en.wikipedia.org/wiki/File:Cfb_encryption.png

[6] http://en.wikipedia.org/wiki/File:Ofb_encryption.png

[7] http://en.wikipedia.org/wiki/File:Ctr_encryption.png

[8] Cryptography and Network Security Principles and practices.

[9] http://www.remobjects.com/

[10] Felix Freiling class lecture of “"Protocols and Security for Wireless Sensor Actor Networks”, University of Mannheim, Germany, March 2008
http://pi1.informatik.uni-mannheim.de/filepool/presentations/attacker-models.pdf

[11] http://msdn.microsoft.com/en-us/magazine/cc700340.aspx?ppud=4

[12] http://msdn.microsoft.com/en-us/library/bb399572%28v=vs.90%29.aspx

[13]http://msdn.microsoft.com/en-us/library/ff650316.aspx

[14]Lea, Doug. Concurrent Programming in Java, Second Edition. Addison-Wesley, 1999

[15] http://msdn.microsoft.com/en-us/library/ms750478.aspx

[16]Turbo Lockbox web page
http://lockbox.seanbdurkin.id.au/HomePage

APPENDIX A

List of Acronyms

AES	Advanced Encryption Standard
BLL	Business Logic Layer
CBC	Cipher Block Chaining
CRUD	**C**reate/**R**ead/**U**pdate/**D**elete
DAL	Data Access Layer
DoS	Denial of Service
ECB	Electronic Code Book
SAP	Service Access Point
SHA256	Secure Hash Algorithm with 256 digest
IDE	Integrated Development Environment
WCF	Windows Communication Foundation
XAML	eXtensible Application Markup Language
XML	eXtensible Markup Language