Software licensing and attendant technologies manage the creation, use, and distribution of software, and also allow for the administration of the resource and the revenue stream from exploitation of the product where appropriate. Although a software license is simply an agreement forming a legal relationship between the user of the software and the vendor, it is a vital part in the evolution of the software idea to market product. Software developers are always seeking new ways to improve the licensing mechanisms that are currently used. The diversity of the business models drives the need for different types of software licenses. In this paper we present a pattern language for software licensing: a set of patterns that can be used by novice or senior programmers to design and integrate an appropriate license type into their software products. Each pattern provides a solution to a well-known problem.
# Table of Contents

Introduction .................................................................................................................................................. 3  
Overview ...................................................................................................................................................... 3  
Summary of the language ............................................................................................................................. 5  
**Pattern 1**: Software License Module ....................................................................................................... 6  
**Pattern 2**: Identity Software License ....................................................................................................... 8  
**Pattern 3**: Multiple Users Software License ............................................................................................ 9  
**Pattern 4**: Time-Based Software License ................................................................................................ 11  
**Pattern 5**: Named User Software License ............................................................................................... 13  
**Pattern 6**: Node-lock (named host) Software License ............................................................................. 15  
**Pattern 7**: Capacity Software License .................................................................................................... 17  
**Pattern 8**: Concurrent Software License ............................................................................................... 19  
**Pattern 9**: Consumptive Software License ............................................................................................ 21  
**Pattern 10**: Cumulative Software License .............................................................................................. 23  
**Pattern 11**: Feature-based Software License ......................................................................................... 25  
**Pattern 12**: Shrinkwrap Software License ............................................................................................... 27  
**Pattern 13**: Evaluation Software License .............................................................................................. 29  
**Pattern 14**: Suite/Bundle Software License ............................................................................................ 31  
**Pattern 15**: Peak Software License ........................................................................................................ 33  
**Pattern 16**: Component Software License ............................................................................................ 35  
**Pattern 17**: Disaster-Recovery Software License .................................................................................... 36  
**Pattern 18**: Open Source Software License ............................................................................................ 38  
Conclusion .................................................................................................................................................... 40  
References .................................................................................................................................................... 40
Introduction

The value of software to customers and computing environments continue to change in a manner that fuels the need for more flexible licensing practices and pricing methods. Typically, every software product is associated with a license that outlines the terms and conditions for using that code. With off-the-shelf packaged products, the user is expected to read (or at least abide by) the license agreement before completing software installation. With tailor-made enterprise systems, depending on the type of installation and software component, licensing is usually agreed before ‘purchase’ and is part of a larger ‘signed’ agreement, ie integrated into the purchase contract.[1] Unfortunately the number of illegal copies on the market, both in consumer and enterprise environments, and the number of unauthorized users (given the moniker ‘software pirates’) is very high. The internationally focused Business Software Alliance reports that on average one in every four software programs currently in use is an illegal copy.[2] To prevent illegal use and copying of software products, licensing modules usually provide additional protection by using the most appropriate encryption algorithms available. Over the past decade software programmers and marketers have adopted several ways to implement license agreements. Although programmers use a variety of different programming languages and a number of distinct algorithms to develop license manager modules for their software products, the tendency is to follow a basic framework that provides licensing functionality.

This paper presents the recurring solutions (namely patterns) to software licensing problems. We propose general solutions for software license types, each of which is presented in the form of a pattern, and is a pattern language for licensing software.[3]

Overview

A pattern language is a way of expressing multiple solutions to complex problems. It is the “sequential and organic structuring of patterns for a specific application domain” that “becomes the method for the development process”. Each pattern within the language solves a very specific problem. Larman[5] points out, that the term “new pattern” could be considered an oxymoron since the term “pattern” describes a repetitive form, although a particular expression of the pattern can be considered new. Each pattern language brings together expert knowledge in a given area as a single expression. Most software developers do not solve application design problems from first principles, but rather reuse tried and tested solutions.[6] Patterns provide such a basis in the development process.

Here we present a pattern language for software licensing, illustrating the most common forms of software licensing, as well as less common ones that may prove useful in certain circumstances. The substantial growth in the development of pattern languages in computer science in the last decade has produced many collections of patterns. Most other work has focused on the object oriented environment and substantial progress in showing patterns in the field of software design and software architecture.[7] Despite these developments, software security is one of the few computer science fields that has not developed a distinct pattern language, although there are several individual patterns widely used.[8] This remains an area with many unsolved problems. Software protection through the mechanism of licensing is ubiquitous and almost every software product has
a license attached to it: this resulted in our attempt to document the pattern language for software licensing.

Software programmers look for better ways to license software in ways that avoid unnecessary licensing complexity,[9] whilst at the same time software publishers struggle to satisfy growing needs of their customers. The current computing environment demands that the licensing system must interact smoothly with other business processes, such as financial planning, impose minimal transaction cost, and meet customer market demands, giving customers flexibility in their use of the software products. It is a difficult task to develop a successful licensing solution for each software solution. Providing a software license requires not only technical development skills, but also the consideration of business operations such as marketing, sales, customer support etc. that play an equally important role. Figure 1 shows the typical software preparation process before it is delivered to the customer.

![Software preparation process diagram](image)

**Figure 1. Software preparation process**

Effective software licensing involves maintaining control over the use of the developed code, whether this is open source, free, or for profit. This may require tracking software usage and identification of the user to protect software against an unauthorized use (see: Software License Pattern). To meet these aims, many software vendors add a license manager module to the packaged software. There are many different types of software license, but selection of the license used is a business decision limited by implementation constraints whilst at the same time taking into account customer software usage patterns and the level of customer resistance to vendor intrusion into software use. However, to date such license management has predominantly been in limited and simple forms, despite the need for changing licensing practices due to customer pressure for flexible licensing, such as making the software package cost correspond to actual use. An example of this can be seen in the movement from the node-locked license type to concurrent use licenses in the UNIX market, which was fueled by the pressure from product users.[10]

The evolution and market acceptance of network technologies has also created a base for networked license models such as concurrent, site, or capacity license types. Unfortunately, there is no perfect licensing solution for every software product. Many times one software application is licensed in number of different ways for different versions. For example, Acrobat Reader is freely available to users under Adobe’s license,
but for someone to use the encoding features requires another license, at a cost, from the company.

Konary et al. point out that 75% of the revenue of surveyed software vendors comes from selling perpetual software licenses rather than the usage based ones.[11, 12] The report also shows that evolution in business processes drive the software protection market towards inevitable change. The study concludes that both software vendors and customers agree that in the near future software licensing will support software leasing and software subscription, with the market trend being towards greater adoption of the latter model.[13]

In order to understand the licensing techniques, it is critical to recognize that the types of licensing policies in use are currently based on a small set of simpler license types. Despite the growing trend to move beyond traditional technologies, the stand-alone licensing methods are still most popular with license managers. The following sections address eighteen common licensing types and show their patterns in the traditional pattern format: a statement of the problem, a description of the context and a solution describing the pattern.

**Summary of the language**

In order to understand the licensing techniques used, it is necessary to be familiar with the types of licensing policies used by software developers. Although there are countless varieties of licensing paradigms, most of them employ variations and combinations from a smaller set of simpler license types. Here we attempt to present the basic software licensing patterns. Over the years the programming experts developed and selected the best ways to solve the licensing problems. We concentrated our research on documenting the licensing practices which proved to be useful over a span of the last decade. (See: Figure 2)

The following are the license patterns:

Pattern 1: Software License Module
Pattern 2: Identity Software License
Pattern 3: Multiple Users Software License
Pattern 4: Time- Based Software License
Pattern 5: Named User Software License
Pattern 6: Node-lock (named host) Software License
Pattern 7: Capacity Software License
Pattern 8: Concurrent Software License
Pattern 9: Consumptive Software License
Pattern 10: Cumulative Software License
Pattern 11: Feature- based Software License
Pattern 12: Shrinkwrap Software License
Pattern 13: Evaluation Software License
Pattern 14: Suite/Bundle Software License
Pattern 15: Peak Software License
Pattern 16: Component Software License
Pattern 17: Disaster- Recovery Software License
Pattern 18: Open Source Software License
Figure 2: Software Licensing Patterns Language Diagram

Pattern 1: Software License Module

Context:
There is a need in software development is to ensure that the rights of creators, intellectual property owners, and other stakeholders are represented concurrently with use or further development of the code. The software piracy problem for software vendors fuels the efforts to improve license management techniques to provide an adequate software copy-protection. To adequately protect software it is common to provide a separate licensing module that manages its use.

Problem/Forces:
Objective: To provide management of the software resource.

Software producers must deal with multiple products targeted at number of different types of customers. In order to satisfy the diverse business goals there is a need for multiple licensing models. The growing demand complexity in computing environment drives the need for a licensing model that satisfies the following:

- is easy to develop and implement
- can be understood by customers
- is flexible
- independent of product or operating platform
- independent of the vendor
- cost effective
- allows for improvements in technology

SOFTWARE LICENSING MODULE provides the means, whether implemented or not, to protect the software from unauthorized use. In this pattern language (see: Figure 3) it
serves as a parent pattern which can be identified as a base for all types of software licenses.

Figure 3: Software License Module

Figure 4. Sequence diagram for Software Licensing Module

Solution:
The SOFTWARE LICENSING MODULE is usually implemented as one of the licensing patterns. The type of license has to be determined by the programmer before the licensing software is developed. The programmer follows the steps of the license design explained in license patterns according to the license requirements.
Pattern 2: Identity Software License

Context:
Restrict the authorized use of software to a specified user or to a specific machine. This is useful when a vendor needs to specify the hardware component that can be used to run an application. The process of assigning the software to a specific identity is called locking.

Problem/Forces:
Objective: Restrict a software license to a specific identity.

This licensing model should meet the following requirements:
- user or machine has a specific identification
- not transferable to another identity
- easy to access by authorized user
- effective cost both to the customer and the vendor

Figure 5. Identity License pattern diagram
**Pattern 3: Multiple Users Software License**

**Context:**
It is very common in today’s networked environments that multiple users need to use the same application. It is beneficial for a company that a predefined number of software licenses be available at all times. It provides an opportunity for the vendor to gather single licenses into one group and put the restriction on the whole group.

**Problem/Forces:**

*Objective:* To allow multiple users to use the software without providing a license to each individual.

This type of license may include a minimum or maximum number of users permitted per request. For instance, a CONCURENT license may require that not more than 15 users can run one application at the same time.
Solution:
To restrict the number of simultaneous users of a software product, there is the need for a MULTIPLE_USERS Software License. The MULTIPLE_USERS LICENSING model is usually implemented as CONCURRENT, CONSUMPTIVE, CUMULATIVE, SHRINKWRAP, NODE_LOCKED software license.
**Pattern 4: Time-Based Software License**

**Context:**
Sometimes there is a need to have an application that should be used only for a specified period of time. It is important to prevent the customer from running the software after the agreed period of time.

**Problem/Forces:**

*Objective:* To restrict the use of software to a defined time period.

To meet customer demand, software vendors often avail themselves of one of the two kinds of license:
- the license terminates on a particular calendar date which is also called the license expiry date
- the license runs for a particular length of time, such as number of hours or days.

For instance, *Sentinel LM* offers the licenses based on a predetermined number of days; starting on the first day that the software is executed.[14] It can be also used in a setting where company needs specific software for a limited period. For example a manufacturer might temporally need a computer application to accomplish a design or to test a new product, (for example, *Xilinx* products might include a time restricted license that is valid for the period of one year).[15]

There are several benefits of using this kind of license;
- reduces cost to customer
- gives customer the opportunity to experience the software “look and feel”
- easy to manage for any number of employees
- vendor benefits from satisfied customers

---

**Figure 9. Time-based license pattern diagram**

might temporally need a computer application to accomplish a design or to test a new product, (for example, *Xilinx* products might include a time restricted license that is valid for the period of one year).[15]
Companies purchase a TIME_BASED software license because this kind of licensing is usually cheaper in the short term than one of the more permanent license types. One drawback to this type of licensing from the point of the vendor is that the user might not stop using the software after the expiry date or when predefined period of time ends.

Figure 10: Sequence diagram for time-based licensing system

Solution:

TIME_BASED software license grants a user the right to use the specified software for a predefined period. This type of license is typically used to support software evaluation. When the application is first installed it should make an entry in the system’s registry providing the necessary information about the time restrictions on the operation of the software. The user should not be able to run the software after the expiry date. This can be effected by implementing a “time-bomb” into the software: code is written into the program wherein after a certain number of days or when certain date is reached the software will no longer work. The time-bomb could be deactivated if the vendor enables a system for extended licensing, such as providing the user with a valid license key that will implement some other licensing model.[16] The License Manager should call (see: Figure 9) on the time checker, which in turn will check the clock for the current time/date that will be compared with the entry made in the System’s License Table. The system will release the license to the user to operate the product so long as the result that has been returned by the time checker is compliant with the license. The License Manager should provide the functionality needed to monitor the time of software usage and/or keep track of the expiry date so that the license can be enforced. Another problem is that using the system date is not always reliable. The user can change the system clock before or after installation. To prevent such fraudulent use of software the License Manager can use an off-site clock.
**Pattern 5: Named User Software License**

**Context:**
Sometimes in a setting where software is IDENTITY_BASED (e.g. e-mail applications or business transaction applications), it is useful to have a mechanism that restricts the use of software to one person. It is typically used in the environments in which “lending” the user-id to someone else conflicts with the very principle of the application or environment.

**Problem/Forces:**
- **Objective:** To restrict a software license to a specific user.

Having the license designated to one user offers a number of benefits, such as:

- uninterrupted application access to all licensed users,
- significantly lower costs that other license types,
- if installed on the network, user can access a software from any workstation,
- can be used to restrict the unauthorized persons from running data sensitive applications.

The fact that this kind of license is not computer dependant (i.e. the software can be installed in multiple locations) is very beneficial in some environments. For example: an architect can have the software installed on his office computer as well as on the personal laptop. Several major licensing software vendors such as: Tarantella [17], FLEXlm [18] and Borland [19] support the use of licenses issued for a specified name.

The drawback for this licensing model is that in a business environment a company will have to purchase for the person, rather than the role that the person fills, and if the eligible user is on vacation, another staff member is not allowed to operate the software. On the other hand, having this kind of licenses simplifies license management.
Figure 11. Named User license pattern diagram

Figure 12. Sequence diagram for Named User licensing system

Solution: This license type is also known as a personal or exclusive license. The NAMED_USER license model permits the access to the software to a specific individual.
To enforce a NAMED_USER software license, the licensing module includes the name of the exclusive user. When the product is first installed, the license program writes user’s identification name into the system’s registry, and this user has the exclusive rights to operate the application. When the program starts, the license manager asks the user to enter his/her name (or an identification sequence). The name checker compares the user’s name to the entry in the system’s registry and returns the result to the license manager. Then the license manager (see: Figure 11) grants permission to the user accordingly. In some business environments, there might be a number of eligible users for the same software that is multiuser. In this case the names of all eligible users are written into the license list of eligible users. If the name of a requesting person is in the license table list (license registry) than the system grants the license to that user.

**Pattern 6: Node-lock (named host) Software License**

**Context:**
It is desirable to have software that is machine dependant. It may not matter who is using the application or how many users are there, but it is crucial that the application is installed and runs on a specific computer.

**Problem/Forces:**

*Objective:* There is a need to restrict the use of software to a specific piece of hardware.

This kind of license allows the use of software on the particular machine (or at least a machine with a specified network identity) for which the license was created. It may allow an unlimited number of simultaneous users of the licensed application on that computer. In this model, users can remotely login to the licensed computer to run the application.

The administrator has to specify the machine identification in order to lock the license to that computer. The problem arrises when the machine becomes outdated and new hardware is needed, when often a new license is needed. This licensing model is also found in computationally intense applications and with software used on workstations dedicated to a particular application. It is difficult to agree on how to identify a machine because some applications might be CPU specific while others might rely heavily on the disk storage space.
Figure 13. Node-locked license pattern diagram

Solution:
If the software is location dependant (e.g. a standalone information kiosk or vending machine) then a NODE_LOCK license should be used. The advantage of NODE_LOCK license is that it is easy to instal. In order to construct a NODE_LOCK license, a license developer needs to provide a set of parameters that uniquely identifies that computer within the software license module. On installation, identifiers from the hardware are written into the executable object file, which can then only be run on that particular hardware. Sometimes installation requires registration of hardware identifiers to exclude reinstallation on another machine. If the hardware identification numbers are
the same as the ones included in the license then the license is granted to use the application on that particular machine. It is a common practice to use hardware signature (fingerprint) as the identification for the computer. The computer's fingerprint is typically formed from a combination of identification numbers of certain hardware components such as processor number, hard drive id, network card type, etc… combined with a crypto hash function. It can be created by the licensing module of the application and should be stored (written into) the system’s licensing registry. A NODE_LOCK license enables the software to run on the designated computer only. Despite the difficulties, NODE_LOCK license is known to be a good solution to the above problem and has been a very popular license type for both off-the-shelf and tailor-made solutions.

Pattern 7: Capacity Software License

Context:
It is often needed to restrict the use of software to some characteristics of hardware. Sometimes it is useful to specify a component of the computer which can be used to run an application.

Problem/Forces:
Objective: To restrict the use of software to the characteristics of the machine where that software is executed.

This type of license is a MULTIPLE_USERS or IDENTITY_BASED license type. It restricts the use of software to certain characteristics of the computer system on which the application is running. For example, these restrictions can be based on the maximum number of processors on which software can run or on the maximum size of the physical memory. A license can be granted for use on less than a full capacity of the machine. For instance, if the processor is divided (partitioned) into smaller pieces, then the license can be issued for the use on one of the partitions. Another type of the capacity-based license is when charges are based on the size of specifically identified resources, such as amount of memory used, number of gigabytes managed, etc. It is common that the license can be based on number of Million Instructions per Second (MIPS) where number of processors used is not limited.
The users with this type of software licenses may have software costs go up or down based on the types of applications running on the machines being consolidated. For example, if the software is of the same type from the same vendor (i.e. infrastructure applications, such as mail or file/print applications), it may be possible to save money by consolidating to less overall capacity at a higher utilization level. Sun Microsystems provides these types of software licenses to its customers for Sun StorEdge Availability Suite 3.0 L10N.[20]
Figure 15. Capacity based license diagram

Figure 16. Sequence diagram for Capacity licensing system

Solution:
If there is a need to assign a license to only a part of the full capacity of the computer, or base restrictions on hardware capacity then the CAPACITY software license should be used. The predefined number of allowed capacity units has to be written into the license.
application. The license application developer has to implement an internal mechanism which will measure the resources used and resources requested. It can be best done with the License Use Management library which is embedded into the license module application and used to calculate the number of capacity units which are used by the computing environment and which are needed for the licensed application. The License Manager compares the available capacity of the operating environment (returned by the Resources Use Manager) along with the parameters in the predefined System License Table. (see: Figure 15) The license is granted only if the available capacity of the machine does not exceed the capacity units specified in the System License Table.

**Pattern 8: Concurrent Software License**

**Context:**
It is very common in networked environments, especially client/server based systems, to have multiple users requiring use of the same application, to engage a concurrent license. It is beneficial for a company that a predefined number of software licenses be available at all times and satisfies the business demand for centralized license contracts.

**Problem/Forces:**
- **Objective:** To restrict the use of software to a defined number of concurrent users.

This software license model is of a **MULTIPLE USERS** type and is most useful in network environments where the number of simultaneous users can use the same application at the same time. There are number of benefits with this kind of license:
- reduces software cost to the company (management/maintenance)
- improves IT planning (demand based on usage measured, predictable intermittent use of subset of employees)
- increases accuracy of license records (due to planning)
- easy to manage by licensing server (central control over server side software)
- employees can change departments (new users might be introduced with no modifications to the system)
- vendor is able to easily control contract revenue
- user company has no control over the number of concurrent licenses (has to comply with the number of licenses assigned at the time of signing the contract)

When this licensing arrangement is used, software can be installed on any number of computers but the actual usage of an application can not exceed the number of licenses provided. Anyone authorized on the network is able to obtain the license provided that one is available. When the license is released by one person it can be reused by another. This is useful especially where the users change seat or department. A typical use of these kinds of license is for a product with relatively expensive licenses that each user will use only part of the time. Companies such as Microsoft, IBM, Sun Microsystems and many other software vendors provide support for this kind of licensing model. Concurrent usage is restricted to networked environments, where it can significantly lower the cost of software. An example of such software can be a word processing application installed on a financial institution’s network where the very principle business lies in using the spreadsheet based software, but word processing activities are required on an ad hoc basis.
The drawback to this licensing model is that when all licenses are in use, potentially important work may need to be deferred (not allowed to be processed).

**Figure 17. Concurrent software license pattern diagram**

**Figure 18. Sequence diagram for Concurrent licensing system**

**Solution:**
The License Manager is responsible for the distribution of the software licenses. The number of the licenses allowed for the concurrent use should be indicated in the
License Manager data table. Every time a user requests a license one should be issued to him/her if and only if the number of the licenses in use does not exceed the number of licenses allowed. Licenses are issued on individual requests until the total number of purchased licenses is exhausted. The License Manager uses the log file to keep track of all licenses in use and if all licenses are in use the user’s request to run the application is denied. In case the license is granted to the user, the License Manager records it accordingly in the log file and increments an internal currently-used licenses counter.

**Pattern 9: Consumptive Software License**

**Context:**
A company’s computing environment today is most likely to include a network build out of high performance desktop computers. It is desirable in such setting to have software licenses that allow multiple users to run the same application. Once the license is used up, it can not be used again.

**Problem: /Forces:**

*Objective:* To provide a number software licenses to specified number of users.

The principle of this type of software license is that a license used once cannot be retrieved or used again. [21] It is the MULTIPLE_USERS type. The licensed software cannot be further used once the number of allowed uses is exceeded.

**Benefits of having this license type are:**

- can be shared by multiple users
- can be used for vendor’s promotions
- can be used for the purpose of the demonstration of software
- proved useful for try-and-buy scenario where the client might choose to try the software product for the predefined number of times before the purchase decision is made
- this license type might be useful in the environment where the number of potential software uses is not known in advance

For example a company might buy 10 licenses, and base the next purchase on the software usage factors. This kind of software license can be combined with other types of licensing models such as CONCURENT LICENSE TYPE, TIMR_BASED LICENSE, NODE_LOCKED LICENSE Pattern …. A problem with the this license is that when the number of executions proved by the license is exhausted the consumer has to contact the vendor to purchase additional licenses.
Solution:

This licensing model has to include the license usage module that will be responsible for recording the number of licenses used. The license developer has to write the Consumed License Counter function that will count the number of licenses consumed and record it into the licensing system’s database. When the License Manager receives a license request the system checks if one is available. The Consumed License Counter should be evoked every time the system receives the request to use the software product.
and when a license is granted to the user. The license should be marked as used and returned to the license server when the application terminates or when the computer system re-boots.

**Pattern 10: Cumulative Software License**

**Context:**
It is becoming common in today’s service oriented computing environments to let the customer to use the software and calculate the payments based on the actual usage. This type of license is especially beneficial to the client/vendor web-services setting.

**Problem/Forces:**

*Objective:* To provide a software license that will allow you to know how many times the users run the application.

This type of license is a MULTIPLE_USERS type and it calculates the number of used licenses and records that for later billing. This type of licensing allows anonymous and named type of software licenses. This type of licensing is similar to the CONCURENT and CONSUMPTIVE license types. The major difference lies in its ability to count and record software usage to be used for future charging.

Many companies prefer to buy CUMULATIVE license for the software. This licensing model allows for paying the fees which are based on an actual product usage and is useful in the post-pay situation. The fee is calculated after the software usage (similar to the utility company – we pay for what we use). This situation is beneficial especially when there is a high cost of the license and there is limited demand for the software in the company. For instance a financial institution might agree to cumulative license for the graphic design software which has been rarely used to edit monthly flyer design. In this case the other types of licenses for this software could be very expensive while paying for the actual usage of it can bring lower costs.

**The benefits:**
- cost effective to the customer
- easy to manage by the vendor
- customer can use as many licenses as he/she actually needs
- employees can change working place (not people dependant)
- minimal IT planning on the customer side

Some businesses choose this type of licensing because they can not afford buying the unlimited use license for given software which usually is more expensive. It is easier on the company’s budget to make periodical payments.

**Problem:**
In this licensing model the company has to maintain continuous relationship with the software vendor which becomes problematic when vendor goes out of business.
Figure 21. Cumulative software license pattern diagram

Figure 22. Sequence diagram for Cumulative licensing system

Solution:
In order to implement CUMULATIVE software license the programmer has to include a separate module in a software licensing system which will be responsible for counting and recording the number of executions or the time units when the software is used. If the vendor uses the number of executions as the base for the billing system then the counter has to be implemented such it counts the number of executions and it records it into the database file where the information about the total usage is kept. If the vendor...
specifies the time units to be used as the base for the payment then the database file will hold the number of total time units used since the last billing. All the information about software execution and the usage report has to be available to the vendor for the reading.

**Pattern 11: Feature-based Software License**

**Context:**
It is beneficial to some vendors (as well as to the users) to license separate features or groups of features within the same software suite. For example, this kind of licensing can be used to allow the user to evaluate the software with restricted features (such as the inability to save files) before deciding to purchase the unlimited access to the application.

**Problem/Forces:**

*Objective:* To restrict the software use to the pre-defined set of functions.

This software licensing model is COMPONENT LICENSE type and it allows the user to use certain features from the main application. Such licensing can also bring some monetary savings to the company. Let say that an accountant is permitted to enter the transactions into the spreadsheet database, but he is not allowed to generate any financial reports. It is a very common situation that a large enterprise has data entry personnel responsible for recording the transactions and a separate budget department personnel that is responsible for generating and analyzing the financial reports. The company does not have to pay for the full license. It can save the money by buying the licenses only for the sets of features of an application that are needed in the particular environment.

With this licensing model is that a user may need a specific feature of an application but does not have a license for that feature without contacting the vendor to obtain an additional license.
Figure 23. Feature-based software license pattern diagram

Figure 24. Sequence diagram for Feature-based licensing system

Solution:
The licensing module developer has to specify a list of features (functions) of the application that can be locked and unlocked for the user. This licensing model has to contain a detailed description about the availability of the features. The function locking mechanism has to be implemented for each function from the list in the License Manager. When the user requests to run an application, the Licensing Manager contacts the Application Functionality Manager module, which in turn contacts the application’s database and unlocks the requested functions.
Pattern 12: Shrinkwrap Software License

Context:
In some situations, particularly for consumer software packages, it is not very practical to get the user to physically sign the license agreement with the vendor in person. In this case the vendor places the license in the package along with the software and documentation.

Problem/Forces:
Objective: To restrict the software use to one computer system.

This type of license is one of NODE_LOCK license type. For instance, when you purchase a copy of WordPerfect, you will notice that the packaging of the software, or the envelope inside the box, have license agreement printed on them, along with a provision stating that opening the envelope or box (or using the software) constitutes acceptance of the terms and conditions of the license. These licenses are generically known as End-User License Agreements (or EULAs), although this term could be applied to any license type. The box contains a serial number that is unique for every copy of software. This kind of license permits the user to run an application on a single computer only.

The drawback to this licensing model is the fact that there is very little to prevent a potential user from copying the software and giving it or selling it to another person.

Figure 25. Shrinkwrap software license pattern diagram
Figure 26. Sequence diagram for Shrinkwrap licensing system

Solution:
The SHRINKWRAP licenses derive their name from the fact that they were initially placed under the shrinkwrap on the outside of software boxes. SHRINKWRAP licenses have been typically used in mass marketed software. When the application is first installed it should write an application key (also known as License key or Serial Number) into the system’s registry. Every time the user requests an operation of the software the license manager should call upon the system registry database and check if the serial number for that application is correct. The license module developer should also include functionality so that the computer’s hardware identification number is read and recorded (it can be a single number representing one of the major hardware or a combination of id numbers of the system’s hardware to form a fingerprint for the system) along with the valid license key of the software upon the software installation. This can prevent running the software by other members of the network (if the computer is part of the network). The License Manager should call (see: Figure 25) on the License Key Checker, which in turn will check the system’s registry for the License Key entry which will be compared with the entry made in the software activation module. The Licensing System will release the license to the user to operate the software only if both entries are the same.
Pattern 13: Evaluation Software License

Context:
Many software vendors let their customers try software before they enter into some other kind of license.

Problem/Forces:
Objective: To provide ‘try-before-you-buy’ software license.

This kind of license, also known as a Demo license, can be based on the predefined time period (see: TIME_BASED pattern) or it can be valid for a number of executions (see: CONSUMPTIVE license pattern). It is very common to provide this licensing model along with other type of license. For instance, the software product has been installed at a school on the local area network and ten concurrent licenses were issued combined with the 30-day evaluation license. In this case ten different users might run the application simultaneously for the period of time not longer than 30 days. Sometimes the evaluation license might be restricted to a limited set of functions (also called feature-based or demo software license). A good consumer example is McAfee Antivirus Application [22] that allows for the free scanning of the computer, but in order to remove the virus infected files one must purchase the virus removal programs. Another example can be a word processing program that has been designed to unlock a time-sensitive or limited-feature version of the program after a user registers his identity with the vendor.

Benefits:
- user has a chance to evaluate the software before he/she decides to purchase it
- minimizes the cost of software to the customer
- easy to design and install
- can be easily combined with other types to restrict the software usage even further

In such a scenario EVALUATION SOFTWARE LICENSE is the best solution.
Figure 27. Evaluation software license pattern diagram

Figure 28. Sequence diagram for Evaluation licensing system

Solution:
The EVALUATION Licensing Model is usually implemented as TIME_BASED, FEATURE_BASED or CONSUMPTIVE License. The type of license has to be known to the programmer before the licensing software development. The programmer follows the steps of the license design explained in TIME_BASED license, CONSUMPTIVE license or FEATURE_BASED license patterns accordingly to the license requirements.
Pattern 14: Suite/Bundle Software License

Context:
It is very common in today’s computing environment to treat set of software application as a unique entity. The vendors usually offer bundle licensing model to provide discounts to the enterprises.

Problem/Forces:
Objective: To restrict the use of two or more products which are licensed individually to the limited number of concurrent uses?

This type of software license is of a COMPONENT LICENSE type and is beneficial in the settings where it is easier for system administrator to manage the number of licenses for a set of individual software as one unit. Typically there are two or more bundles on the system and the system manager can view all the bundles in a licensing module. For instance the company uses a number of Microsoft products as well as several Sun Microsystems applications. In this case the licenses can be bundled into two groups according to the vendor that produced them. The system administrator can delete the bundle from the system but deleting the single license is not allowed. It is possible to check if all the licenses in the bundle have been enrolled and to verify the consistency of a bundle. For example if a user was licensed for a bundle of five products and 100 uses, when 30 of product A are in use and 20 of product B are in use then the customer has seventy uses of product A left and eighty uses of product B. He also has a hundred of uses on each of the three remaining products. The SUITE_BUNDLE Licensing Model can be derived from the CONCURENT license type (see: Pattern 8).
**Figure 29. Suite/Bundle software license pattern diagram**

**Figure 30. Sequence diagram for Suite/Bundle licensing system**

**Solution:**

In the SUITE_BUNDLE licensing model each product is still represented by its own license agreement. In addition to that there is a higher level of the usage authorization which is represented by the SUITE_BUNDLE license. The license module developer should link the products in the bundle by a common license key and the password. It is possible to represent all the products in the system by a single bundle license. The number of the available licenses should be indicated in the License Manager’s module. When the user requests to run an application the license should be issued to him only if the number of the licenses in use does not exceed the number of licenses allowed. The log file is used to record the number of licenses in use. The License Manager module is implemented the same way as the CONCURENT Licensing system (see: Pattern 8).
Pattern 15: Peak Software License

Context:
Very often in networked environments, especially client/server based systems, users need to use software more often at certain times of the day. For some applications it is not appropriate to limit the software usage to a set of concurrent licenses.

Problem/Forces:
Objective: To manage software use based on the different time of the day.

This licensing model is used primarily for the billing purposes. A company usually purchases licenses for a lower fixed fee and additionally pays for peak usage. For example it is not in anyone’s interest to deny access to any mission critical application especially in industries such as National Security or Health Care. This licensing model is targeted at the applications to which the users are denied the access during the busiest times. Such situation creates customer service issues and/or customer dissatisfaction. The Peak Licensing model allows users and software vendors to monitor and audit actual product usage. The bill is based on data gathered during different times of the day. The License Manager System generates the usage report logs based on the database file. The reports break down customer usage based on the time of the day. This licensing model appeals chiefly to companies that provide services that experience spikes in demand. Without such a license these companies would have to pay for expensive, powerful systems that they need only during the peak-demand periods. For example HP introduced Pay-Per-Use licensing model that allows it’s customers to pay monthly fee based on the actual usage.[23] Having such licensing system the user can save the money by paying less during calm periods. Typically during all periods the user must pay a fixed monthly fee.
The problem to this licensing model is that the user has to maintain continuous relationship with the vendor.

It is beneficial for a company to pay monthly fee based on the actual usage of the software. In such situation PEAK software license should be issued.

Just as in Pattern 4 (TIME_BASED licensing model) using the system clock is not very reliable. The user can change the system clock before using the application. Using the off-site clock would prevent such fraudulent use of software.
Solution:
The programmer has to include the time checker module in this model. Since the usage price depends on the time of the day the time checker has to be able to recognize and return the exact time meaning hour and minutes values. The time checker has to communicate with the system’s clock when it requests the current time and then it sends that information to be recorded in the usage log file. The user log should include the
functionality to generate the usage reports based on the different time of the day since the usage price is differentiated based on the time of the software use.

**Pattern 16: Component Software License**

**Context:**
Sometimes there is a need to use a specific software component along with an access to different application. If such combination of use is needed then COMPONENT software license is the best suited for such purpose.

**Problem/Forces:**

*Objective:* To restrict the use of software to pre-defined components.

This licensing model governs the use of other licensed runtime software components with the application being developed. For instance the spell checker from software A can be used within word processor from software B. Typically the software product is split into modules so the publisher can create various license types that offer access to different parts of the software in any combination. For example a programmer is developing Visual Basic developer’s suite. The text editor and GUI building tool might be constrained by the basic license agreements. The more expensive version can include UML modeling tool and the top line enterprise edition can enable database binding’s capability. It is important to mention, that this model is very similar to the FEATURE based licensing model (see: Pattern 11). The difference lies in the fact that the FEATURE based model is used to limit the usage of some functions of the software and the COMPONENT based licensing is used to permit the usage of one or more individual components with another application.

*Figure 33. Component based software license pattern diagram*
Figure 34. Sequence diagram for Component based licensing system

Solution:
The development of the COMPONENT_BASED licensing model is possible only in the modular products. These applications have their code organized carefully into separate modules. The licensing software developer should create several licenses that each carry unique license key. Each module should carry a separate license that can be combined with other product’s license. Upon the software installation the licenses’ information should be written into the system’s registry. Any time the user requests to run an application’s component the License Manager should be able to retrieve the licensing information from system’s registry and compare it with the user’s access privileges.

Pattern 17: Disaster-Recovery Software License

Context:
It is desirable that the company needs additional software licenses due to software crashes or hardware malfunction. In this case the DISASTER_RECOVERY is the best solution.

Problem/Forces:
Objective: To manage software use in the disaster-recovery scenario.

This type of software license controls access to the application during the time when the network server is down and there is no other way to activate the application. An example of such situation might be the last summer’s massive power outage in the northeast USA.
and Canada. Many companies found out then that having disaster-recovery plan with copies of the software and valid licenses pays off. There are number of benefits of having such software license:

- prevents loss of revenue for customers in case of software crash
- easy to implement and deploy
- can be designed using other patterns
- there is no need for continuing relationship with the vendor
- can be easily included in customer’s IT plan

There are several types of the disaster-recovery license. It can be implemented as a TIME_BASED, CUMULATIVE or CONSUMPTION based. If license is TIME_BASED it’s usage is restricted to the predefined time period, if it is a CUMULATIVE license type then the license becomes available for number of executions. If the license is CONSUMPTION based type then software usage is recorded into the log files and the company is billed based on the number of software executions.

Figure 35. Disaster-Recovery software license pattern diagram
Solution:
The DISASTER RECOVERY Licensing model is usually implemented as TIME_BASED, CUMULATIVE or CONSUMPTIVE License. The type of license has to be known to the programmer before he can develop the licensing module. The programmer should follow the steps of the license design explained in Pattern 4 (TIME_BASED license), Pattern 9 (CONSUMPTIVE license) or Pattern 10 (CUMULATIVE license) accordingly to the license requirements.

Pattern 18: Open Source Software License

Context:
Commercial software is generally sold for a fee, with restrictions on use, while Open Source Software is available for free. However, “free” does not mean that such a software can be released without a license.

Problem/Forces:
Objective: To manage the use of open source software.

Open source software is software that has its underlying source code freely available to evaluate copy and/or modify and is freely available to the public. The most known example of such application is Linux - an open source version of the UNIX operating system [24]. The Apache Web Server [25] is another example of open source software. Although the software is open to everyone it carries the license with it which defines the rights and obligations that the user has. Typically the open source software license will contain the clauses about protection of moral rights and protection of
openness. For example, some authors insist that any work derived from their software has to be an open source as well. Some licenses require that any redistributor maintains the same license and therefore each user has the same rights no matter if the software was received from the author or from the third party. There are several types of open source software licenses. Some of the most common include: BSD (Berkley Software Distribution), GPL (GNU General Public License), MPL (Mozilla Public License).

<table>
<thead>
<tr>
<th>Software Licensing System</th>
</tr>
</thead>
<tbody>
<tr>
<td>License Manager</td>
</tr>
<tr>
<td>License Repository</td>
</tr>
<tr>
<td>System License Table</td>
</tr>
</tbody>
</table>

Figure 37. Open-Source software license pattern diagram

<table>
<thead>
<tr>
<th>User</th>
<th>:application</th>
<th>:License Manager</th>
<th>:License Table</th>
<th>:License Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>request()</td>
<td>checkLicense()</td>
<td>result := getLicenseType()</td>
<td>read text license()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>allowUsage()</td>
<td></td>
</tr>
<tr>
<td>[positive] execute()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 38. Sequence diagram for Open - Source licensing system

Solution: If you plan on releasing the source code to the public with no charge for it you need an OPEN_SOURCE SOFTWARE license. This type of license is usually
implemented as follows. The text file contains a written version of the license which
upon software installation is recorded into the System License Table. The licensing
module should include the readLicense() function which will be evoked every time the
application starts. The function will read the text version of the license to verify
predefined statements which will identify that software as being available to the public.
The decision to permit the user to run the software should be based on the outcome from
the readLicense() function.

Conclusion

This paper presents some of the key software licensing patterns that occur in
today’s computing environment. To design, develop, and implement a software licensing
system are not trivial tasks. The changes in software market in recent years have
increased the importance of having well designed and flexible software licenses, and
there is growing pressure in the industry from both vendor and consumer to address these
needs.

In this paper we have examined some of the issues that need to be considered
when developing a license manager module that implements various types of software
license. The best way to make sure that one has an adoptable solution is to keep the
development focused on customer demand and vendor need within the constraints of the
legal licensing paradigm. Through constructive guidance with patterns for software
license developers, and by reusing generic licensing patterns, some of the growing
complexity in software licensing may be overcome.

References

Lexis, 2004. In the vast majority of software contracts there is no outright purchase of the
software, nor any traditionally ‘signed’ agreement, though clickwrap may be treated this way.
stated as a problem consists of the title, problem statement, context and solution. Christopher
Alexander is known for the design of the first Pattern Language for the architectural designs in
1977.
[4] Aarsten, A., Brugali, D., Menga, G., ”Designing Concurrent and Distributed Control
Systems”. Communications of the ACM, October 1996, Volume 39, Number 10, pp.51
218
ISBN: 020163612. pp.1
[7] For recent examples, see Proceedings of the 26th International Conference on Software
Engineering (ICSE04) and ACM SIGSOFT Software Engineering Notes Volume 29 , Issue 3
(May 2004) which have several pattern bases papers.
[8] There are few formal pattern papers in this area, such as Sandeep Kumar and Eugene H.
Spafford, A pattern matching model for misuse intrusion detection, Proceedings of the National
Computer Security Conference (1994), 11–21, but intrusion detection often depends on analysis
of attack patterns.
Perpetual licenses are paid for on a one-time basis. The user has a right to use the licenses software so long as that user owns the license—i.e., without time restriction, although typically the vendor will want to sell an upgraded version before too long.

Subscription licenses are paid for with a reoccurring fee to continue using the software. If the fee is not paid, the software stops working.


Time-Bomb solution explained more on Real Basic Developer website: http://www.rbdeveloper.com/browse/1.6/1606/

The Apache Web Server can be defined as software that is used to allow a computer to deliver World Wide Web pages to a computer that requests the page.