Subhakara Valluri

Development and Integration of a Document Management System in a Modular Web Platform with Efficient Storage Mechanism

Master thesis based on the examination and study regulations for
the Master of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner : Prof. Dr.rer.nat Hans-Jürgen Hotop
Second examiner : Prof. Dr.rer.nat Henning Dierks

Day of delivery June 7th 2010

Subhakara Valluri

Title of the Master Thesis

Development and Integration of a Document Management System in a Modular Web Platform with Efficient Storage Mechanism.

Keywords

Cuyahoga, Subversion, NHibernate, Inversion of Control, Document Management System, Role Based Access Control (RBAC).

Abstract

The web based document management system is used for distribution, submission, review and management of documents. Clients can access the documents from anywhere in the world by using instantaneous on-line access. Users submitting documents can update them, reviewers can add comments, and the interface is designed as per client's requirements. In this thesis, document management system is designed and implemented with space-efficient document storage mechanism and version control using Subversion. Additionally a security concept of the system is designed and implemented according to client's specific requirements using Role Based Access Control (RBAC) techniques.

Subhakara Valluri

Thema der Masterarbeit

Entwicklung und Integration eines Dokumenten-Management-System in einer modularen Web-Plattform mit effizienten Speicherungsmechanismen.

Stichworte

Cuyahoga, Subversion, NHibernate, Inversion of Control, Document Management System, Role Based Access Control (RBAC).

Kurzzusammenfassung

Das Web-basierte Dokumenten-Management-System wird für die Verteilung, Einreichung, Überarbeitung und Verwaltung von Dokumenten verwendet. Clients können von der ganzen Welt durch Online-Zugang auf die Dokumente zugreifen. Benutzer können auf ihre Dokumente zugreifen und sie ändern, andere Nutzer können durch Hinzufügen von Kommentaren Dokumente überarbeiten. Die Benutzerschnittstelle ist aufgrund der Kundenanforderungen konzipiert. In dieser Arbeit wurde ein Dokumentenmanagement-System für die Platz sparende Speicherung und Versionsüberprüfung von Dokumenten mittels subversion entwickelt und implementiert. Zusätzlich wurde aufgrund der speziellen Kundenanforderungen mittels Role Based Access Control (RBAC)-Techniken ein Sicherheitskonzept des Systems entwickelt und umgesetzt.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: Introduction

The presence of World Wide Web (WWW) has a great impact on the working style of every organization. The Web can be used as an inexpensive and powerful medium of communications. Due to the fact that Web browsers can be run on any type of computer, electronic information can be accessed consistently and concurrently by all employees irrespective of their locations. The total corporate information like training materials, procedures and directories can be converted to electronic form and made available through the Web. Having a single source for all the information significantly reduces the maintenance costs and has the benefits of information exchange. It also provides a companywide system irrespective of the underlying information technology (1).

Due to the benefits associated by having an enterprise wide information management system. Every company started using a Web based document management system to manage their documents in electronic form. Thus Web based document management system's occupies a point of interest in the current markets. Although there is a wide range of products available in the market, still there exists a gap between company's requirements and the solutions available in the market.

The key aspects that need to be addressed are as follows:

- Often big companies have large number of documents that needs to be available to their internal or external clients worldwide. Due to increase in number of documents and their corresponding versions, there is a huge requirement of the storage space for the documents. Apart from the storage space there is also in need of better techniques for faster access of the documents.

- There is always a mismatch between the company's requirements and the available products in the market. For instance a company wants to have simple Web-based document management tool, but due to the complex products available in the market one needs to forcefully buy them. There is wastage of resources associated with these kinds of situations.

- In any system, security plays a key role for bringing confidence about the information. Every company wants to have a sophisticated security mechanism for their information.

1 Introduction

This document discusses an efficient approach to handle the storage space of any Web-based document management system by eliminating the necessity of storing multiple versions corresponding to the single document. It also discusses the benefits of an effective security mechanism named RBAC (Role Based Access Control) and its applicability to the Web. This thesis also shows an approach to have a well suited application by selecting each and every component according to the specific needs of the company. Apart from showing different approaches it also realizes an end-to-end solution to use.

This thesis is organized as follows. Chapter 2 presents an overview of the background issues required for this topic. Chapter 3 gives a detailed analysis of the requirements. In Chapter 4, the explanation of different technologies used in this project is discussed. Chapter 5 consists of the total design of the web-based document management system. In Chapter 6, the implementation part is shown by having an overview on the user interface of the system. Chapter 7 shows the testing techniques and its applicability to the system. The Chapter 8 gives an overview about the further work. Finally Chapter 9 concludes the work.

CHAPTER 2: Background

2.1 Evolution of Document Management

The invention of photocopying in the 60’s reduced the cost of duplicating information. In the 80’s fax became the popular data transfer method and was used for sending offers, graphics, etc., but not useful for the documents. Thereafter during 80’s, the introduction of personal computing for the day–to-day work helped to reuse information with the use of word-processing and other software. Finally, in the late 80’s and early 90’s Internet made possible the transfer of a document via email which was a great achievement for today’s document management. The evolution tended to join groups of users, professionals, etc., in different spaces which were called Portals or Virtual Communities. Later, companies felt the importance to offer services and improve their contents which led to the development of Electronic Document Management Systems, where documents are stored in a web server and users interacts with this web server generally referred as central repository by the provided web user interface (2).

2.2 Document Management System

Document management is where people spend time and money for the obvious solutions to the problems. A document management system is generally a database in which some records contains or index large files. Besides the documents themselves, DMS also stores data about those documents, which is the main reason building a document management system rather than storing into a shared directory. This additional data is often called *metadata,* because it is data about data. Apart from storage and retrieval of documents, there are two commonly seen features in the document management systems: *version control*, which allows the total history of a document to be captured and tracked and gives the historical versions of the document; and *indexing*, which utilizes various tools to compile information about the location of keywords and phrases within documents to allow searches being performed on their contents (3).

The definition of Document Management System provided by AIIM (4) is as follows: “*Document management, often referred to as Document Management Systems (DMS), is the use of a computer system and software to store, manage and track electronic documents and electronic images of paper based information captured through the use of a document scanner*”.

A well-designed document management system controls the life cycle of the documents in an organization and fits with its culture and goals. It also makes finding and sharing the information with ease. It also provides features at each steps of a document life cycle.

2.3 Content Management System Vs Document Management System

A Document Management, while still recognized and utilized independently, it is typically part of an Enterprise Content Management environment. The definition of Enterprise Content Management System from AIIM (4):”*Enterprise Content Management is the technologies used to Capture Manage, Store, Preserve and Deliver content and documents related to organizational processes”*. A subsection of Content Management is Web Content Management .The definition of Web Content Management from AIIM (4): *“A subsection of Content Management is Web Content Management or WCM. A WCMS is a program that helps in maintaining, controlling, changing and reassembling the content on a web-page*

2.4 Importance of Document Management System

Every company feels the importance of Document Management Systems (DMS) to control their exponentially increasing number of documents. Companies often resist this urge because of costs and complexity involved in the implementing a DMS. Using a DMS effectively, needs a major change in working life cycle. Most of the technical costs can be reduced using the open source data bases and software’s and integrating to the windows environment. A best DMS not just provides access to all documents, but also provides safe access to all internal employees and also clients or other participants of the project via Internet or Extranet.

To increase the productivity of a company, efficient information management is primary requirement. The standard features of DMS includes searching functionality, red-lining, printing and workflows, revision and version control, document security, document linking, status reporting, issue management and remote access. Many of these features saves time, simplify work, protect the investment made in creating these documents, enables audit trail, ensures accountability and enforce quality standards. The figure 1 shows the comprehensive workflow of the document management system.

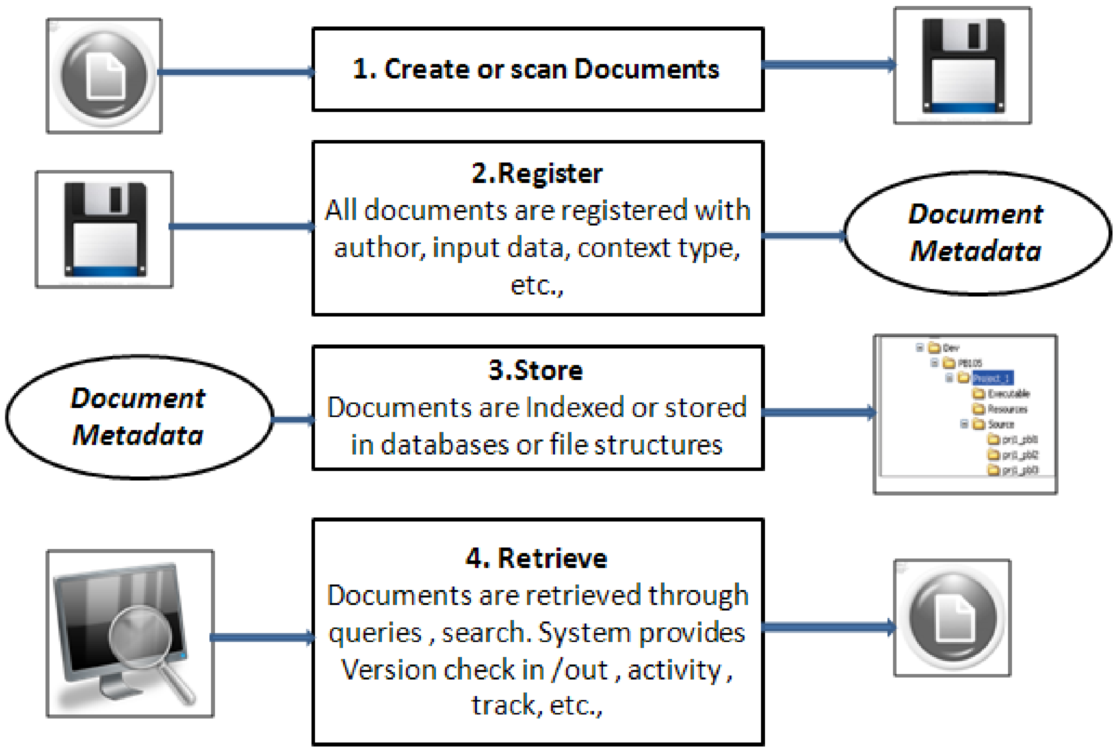

1. Create or scan Documents
2.Register
All documents are registered with author, input data, context type, etc.,
Document Metadata
Document Metadata
3.Store
Documents are Indexed or stored in databases or file structures
4. Retrieve
Documents are retrieved through queries , search. System provides Version check in /out , activity , track, etc.,

Figure 1: DMS Functionalities at different levels
Source: (5)

The EDMS have following advantages (5):

- Efficient location and delivery of documentation
- Ability to manage documents regardless of originating system or format
- Ability to encompass and integrate with the existing computer based system
- Control of access, distribution and modification of documents

- Provisioning tools to edit documents and add markup information whatever the source of document.

2.5 Limitations of Document Management System

There is also teething problems on the other hand (2).

- The technologies and markets are changing fast, so it is better needed to keep up-to-date information about new applications, new vendors, new uses, and new implementation approaches.

- All information must be in electronic format which is created electronically or scanned in from a paper based version. This includes hand written notes, sketches and large drawings .Much of the effort is wasted with the incompatible interfaces especially paper based ones.

- Business and organizational issues such as start-up costs, payback analysis, cost justification and savings must be addressed

2.6 Information Security and Access Control

Access control is a mechanism by which resources of information system is safe guarded. This controls the authority of the information system. It is only part of a total computer security solution. In general risks in the Information security can be classified in to three types, confidentiality, integrity, and availability (6).

- *Confidentiality*: It shows the need for the protection of information safety and making it private. It generally consists of information ranging from financial information to security information such as passwords and user ID's

- *Integrity*: It brings the concept of protecting against the information being altered by unauthorized users, which helps to avoid modification of data by hackers there by having full control on the hands of administrator.

- *Availability*: It means to have required information available for use when needed. There are many attacks which attempts to overload corporate web servers. This can be minimized by availability concept.

2.7 Need for Custom made Solution

Currently there are many different tools available in the market to manage documents. The tools involved are continuously changing or the companies selling them may disappear. This situation always generating costs in terms of updates and the development of import/export scripts and shows that the users do not really have the control on their information storage.

Another scenario arises due to complexity of the tools. Often user needs just a simple tool to manage their documents with minimum features, but the tools available in the market are generally complex with high number of features .It is known fact that number of features comes at the cost of complexity. To have one to one match with the users and their actual needs one needs to develop their own system to reflect their actually work flow.

2.8 Company’s Introduction and Existing Solution

Alpine Electronics Gmbh (7) is one of the leaders in the car audio, mobile electronics and navigation systems. They have global research and development facilities in Asia, Europe and the United States. Alpine is the global leader for in-vehicle navigation systems in Japan, North America and Europe for the aftermarket and OEM factory installations.

Currently Alpine is using I-Notion (8), which is a product lifecycle management portal. I-Notion belongs to company named I-Logix (9), which is leading provider of Model-Driven Development solutions for system design software development focused on real-time embedded applications. In March 2006 I-Logix was acquired by Telelogic AB and integrated as a business unit for embedded modeling. In April 2008 Telelogic AB accepted IBM’s offer for making its products part of IBM’s Rational Software Unit.

The features of I-Notion as stated by its company are as follows, it is a fully scalable and easily deployable PLM solution. Because it is a web-based portal, any computer with Internet Explorer can access I-Notion and there is no need of installing any client. As a central server based system, document submitted are available worldwide without the need of replications. It has a functionality of multi-language support for languages like Chinese, Japanese, German and English (10).

CHAPTER 3: Requirements Analysis

After utilization of I-Notion (8) for some consecutive years, Alpine electronics (7) identified the necessity to change their existing solution for better productivity and maintainability. Due to the limitations of I-Notion they came across many challenging issues that need to be addressed.

3.1 Drawbacks of Existing Solution

- *End-of-Life*: Any software solution to use successfully for longer period, it is always crucial to have a customer support from the solution provider. Due to successive acquisition of the parent company by many other companies results the lack of further customer support. There is no longer marketing, selling or promotion of I-Notion. It is always problematic to work with a solution, which reached end-of-life period.

- *Bad Storage Solution*: Storage space always has very crucial role in the usage of any document management system. Every user wants to have a solution which contains minimum space for the storage. Currently I-Notion is occupying enormous amounts of space for storage due to lack of efficient version management solution. In I-Notion every version is stored as an extra document, which is finally resulting thousands of individual documents. This weak solution is increasing the storage space multiple times.

- *Storage Congestion*: Apart from weak version management solution for document storage, it also has every document in the same directory. This leads to dramatic performance failure in the total system as NTFS[1] performs very badly when there is lot of files in same directory.

- *Lack of control*: There is always beneficial to link different information systems to obtain better productivity. Every system must provide a means for data flow and exchange. I-Notion is quite closed system, where one cannot link with other corporate information from another information system.

- *Lack of Multi Browser Support*: From the beginning I-Notion only supports Internet Explorer[2]. There are quite a lot of browsers for instance Mozilla Firefox[3], which provides better access over internet. I-Notion has also a drawback of supporting only older version Internet Explorer 6 due to some security issue .There is a need of some modifications in order to view on latest versions of Internet Explorer.

[1] http://en.wikipedia.org/wiki/NTFS
[2] http://www.microsoft.com/windows/default.aspx
[3] http://www.mozilla-europe.org/de/firefox/

- *Lack of Extendibility*: There is no chance for further modification of the system according to present working requirements because of its quite closed nature.

Due to all these drawbacks there is not much further options left apart from finding a new solution.

3.2 Further Choices

As the need of new solution is arrived, the choices of options left in front of desk are determined as follows

Choosing different product: Replacing the existing document management solution by the new product from another company is one of the possible solutions. Although it can be carefully selected according to requirements and drawbacks from old system, there is always possibility of facing old problems back again. Apart from this, there is a huge cost involved in purchasing new system.

Having own implementation: The creation of new document management system is one of the possible solutions. It brings the comfort of choosing different components according to the requirements by keeping in mind the drawbacks of old system. In general by carefully planning and monitoring, one can reduce the costs involved in the development. Having an own implementation of system always provides the full control over the system. At the bottom line one need certain time period to carefully plan and develop the new system, as implementing the new system is not ready made.

By carefully comparing both solutions one clearly feels that having an own implementation of document management system is better than buying an existing product from the market.

3.3 Benefits of Selected Choice

- *Ownership*: Owning the tool makes it’s easier to maintain. It is easier to perform any modifications or extensions based on ongoing change in the requirements.

- *Efficient Version Control*: As the previous solution has weak storage mechanism, one can find the best approach to manage versioning and minimize the storage space required.

- *Freedom on components selection*: Because of ownership, one can select any number of required components based on the budget and other constraints. The selection of database is very crucial in performance and maintenance of whole system.
- *Browser compatibility*: The new implementation can be taken care of necessary browser compatibility. From the beginning of development one can keep in mind the probability of using different internet browsing clients and their corresponding security issues.
- *Less expensive*: By careful planning and screening, it is easier to obtain a cheaper solution. It also provides fewer costs for maintenance and future extensions.

3.4 Analysis of Version Control Mechanism

The very first and important aspect of whole development is about finding effective way to control the document versions and storing them. Many possible mechanisms are available on the market according to one's need.

In broad view document management systems includes file systems, (X) HTML[4] and XML[5] repositories. There is wide range of practices for version management of object, XML, HTML, text documents. In general some version management schemes are used to manage versions in different representation models. Normally all these schemes typically assume (11):

- An initial expectation of how versions will be manipulated.
- Constant priorities between storage space usage and average access time.

Apart from there is also an adaptive document management versioning scheme. These are different schemes and their mechanisms are discussed as follows:

Storing Latest version and backward deltas*:* In this scheme, only the latest version and backward deltas are stored. In general backward deltas are difference between current and previous version. In order to generate the previous versions, the backward deltas are

[4] http://www.w3.org/TR/xhtml1/
[5] http://www.w3.org/XML/

necessary. The access to the current version is always faster than the previous version, because it needs to calculate the previous version on the fly. This scheme is very appropriate when one needs very efficient mechanism in terms of limited storage space.

Storing every single version

In this simple scheme each and every version is stored as a separate document. But the main drawback of this scheme is storage overhead. This scheme is useful for whom the primary importance is historical information and its fastest retrieval.

Adaptive versioning

This is an adaptive document version management scheme, which supports different management schemes. In this scheme the storing of document depends on the observation of usage patterns and the utilization level of the document. Every document is categorized into three different levels named as *pertinent*, *relevant* and *obsolete* (11).

- *Pertinent level* shows that document is very important. These kinds of versions must be kept as stored versions, because one needs faster retrieval of them due to regular usage.
- *Relevant level* shows that document is not highly important but it is needed some times. This kind of documents can be maintained as calculated versions, where only deltas are stored and the document is created on fly upon request.
- *Obsolete level* shows that document is no longer in use. Normally one can delete this kind of versions for having better performance and reducing the volume of document history.

This technique is dynamic and continuously monitors and changes the priority levels of document according to usage patterns. One needs an effective way to realize this system.

3.4.1 Access Time Vs Storage Space

The main part of selecting version management solution surrounds between the fastest retrieval of previous versions against using less storage space. According to the requirements it is quite clear to have a solution with efficient storage mechanism, because one needs the current version mostly than previous versions. In case of retrieval of previous version one can admit little slower performance in order to achieve best storage space efficiency.

3.4.2 Analysis of Different Schemes

After observing above different schemes one needs to decide the final scheme depending on several requirements and constraints. The *adaptive versioning* scheme is highly effective but it lacks the prominence in market, so one need to develop it from scratch which may consume quite a lot of development time and costs. *Storing all versions* scheme may brings back the problems associated with the volume of document storage. In the current scenario *Storing latest version and backward deltas* scheme will suits best, because it is highly common practice in market and one can select required versioning engine from the different systems. Another benefit associated with this decision is chance of having some open source systems, which reduces the final costs involved in the development of document management system.

3.4.3 Comparison of Different Versioning Systems

The table 1 clearly shows that CVS[6] is very old-fashioned version control system. It is replaced by Subversion at later times and currently CVS is just maintained without any additions. Team Foundation Server[7] is well maintained and updated by Microsoft. The main drawback of this system is lacking of Linux[8] platform support and very expensive to buy. Clear Case[9] is also actively maintained and developed by IBM Rational by supporting wide variety of operating systems. The main drawback of IBM Rational is due to high costs associated in getting of its license. Mercurial[10] and Subversion (12) are well maintained and actively developed, both of them are free. Subversion has even commercial support upon request. But they use different repository models, one need to decide on the lines of which repository model better suits their needs.

[6] http://www.nongnu.org/cvs/
[7] http://msdn.microsoft.com/en-us/library/ms181232(VS.80).aspx
[8] http://www.linux.org/
[9] http://www-01.ibm.com/software/awdtools/clearcase/
[10] http://mercurial.selenic.com/

Candidates	**Maintainer**	**Development status**	**Repository Model**	**Supported Platforms**	**Costs involved**
CVS	The CVS Team	Just maintained without adding features	Client-server	Unix-like, Windows, Mac OS X	Free
Team Foundation server	Microsoft	Actively developed	Client-server	Server: Windows Server 2003; Clients: Windows and Web included	Non-Free
Mercurial	Matt Mackall	Actively developed	Distributed	Unix-like, Windows, Mac OS X	Free
Clear Case	IBM Rational	Actively developed	Client-Server	Linux, Windows, AIX, Solaris, HP UX, i5/OS, OS/390, z/OS	Non-Free
Subversion	Collabnet Inc	Actively developed	Client-Server	Unix-like, Windows, Mac OS X	Free but commercial type services also available

Table 1: Comparison of Versioning Systems
Source: (13)

3.4.4 Client Server Vs Distributed Repository Models

In its simplest form Client-Server Model is also called as centralized version control. As its name hints centralized versioning model has single place for check in and check out of the code. Distributed versioning model is quite different, where it doesn't have single base to fetch the code, different branches will have different parts of the code. Centralized version control highly focuses on synchronizing, tracking and backing up files. Distributed version control highly focuses on sharing changes.

The figure 2 shows clearly how centralized version control works. Here every user synchronizes and sends the changes to the main trunk. User1 changes must be sent first in order to be seen by other users.

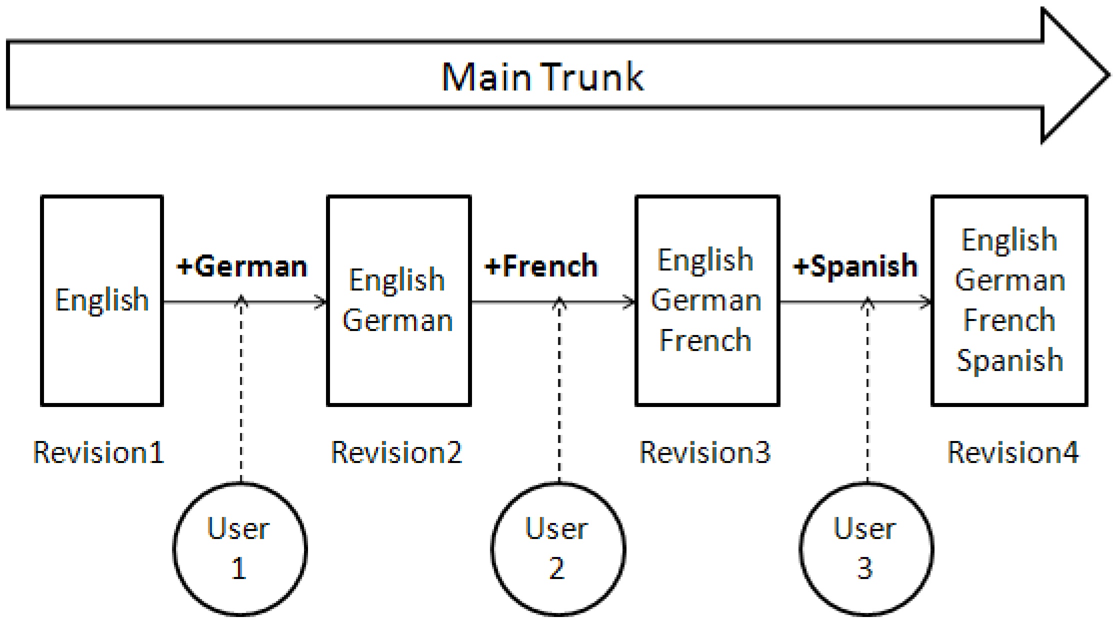


Figure 2: Centralized Version Control System

Source: (14)

The figure 3 shows the distributed versioning model; here every person has its own local repository which one can share with other users. It doesn't mean that there is no main repository, if one desires they can add their local content to the main repository. In the figure 3, it is clearly shown that user1 and user2 is sharing their work and they are also updating the corresponding changes in the main repository. Similarly user2 and user3 shares their content and also send their changes to the main repository for making it available to the other users (14).

The key disadvantage associated with distributed model is that there is not really latest version because anyone can't immediately know which user has latest version. Again one needs central location to obtain the latest version of the document.

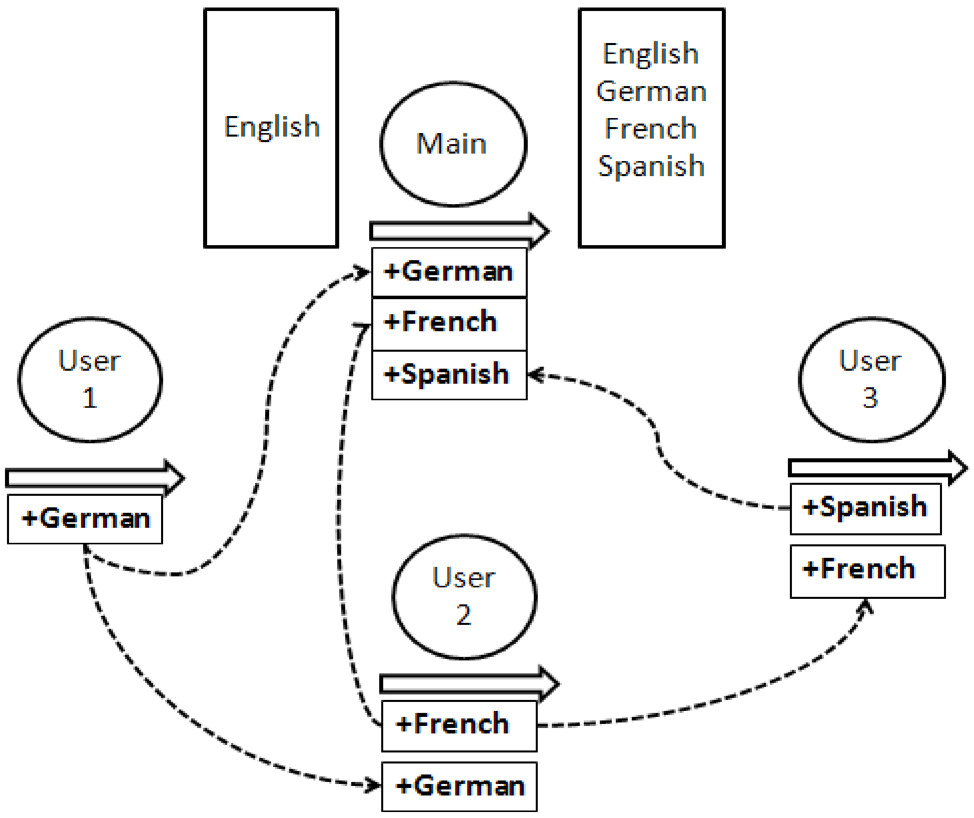


Figure 3: Distributed Version Control System

Source: (14)

From the observation it is quite clear that distributed and centralized version control systems addresses different set of problems. In order to have better selection, one needs to select according to own requirements. It is quite clear from the requirements of the document management system, the centralized repository model is well suited than the distributed repository model because one need a way to immediately recognize latest version. Moreover the use of version control system in this case is not for source code control. If it is for source code control distributed model may be good choice. Before taking the final choice it is better to know which type of files can be versioned using subversion.

Subversion can handle human readable text files, which contains the text in ASCII format. They also can version the binary files, which might be also executable files. Apart from these it can also version the directory files which contains the information about how to access other files. Finally from the observation it is evident that the subversion is better solution for this case (15).

3.5 Analysis of Access Control Techniques

At first user's account credentials and identity are authenticated using Authentication and Identification techniques. There after control over access to system's resources is based on access control mechanism used. So access control plays a key role in the security of the web-based document management system. The analysis of different types of access control techniques is needed in order to select the suitable technique for the web-based document management system.

There are mainly two types of access control techniques to analyze for the better suitability

Role Based Access Control: It corresponds to the level of security to close resemble the organization's structure. Every user is assigned with one or more roles and each role is has one or more privileges. This is clearly shown in figure 4.

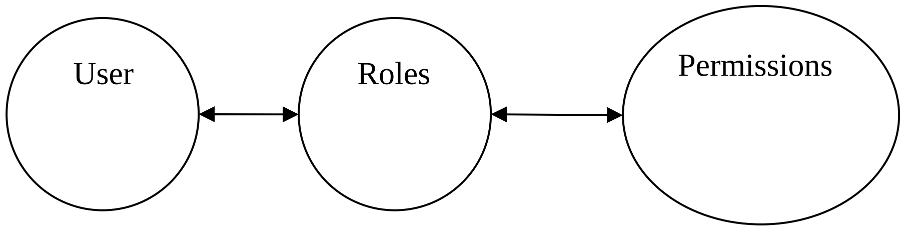


Figure 4: Description of RBAC

Source: (16)

Rule Based Access Control: The grouping of access rights can also be done using rules. Rules are based on conditions having left hand side (LHS) and one or more actions in right hand side (RHS). The actions in RHS are executed only, if the conditions in LHS are satisfied. The simple rule is shown as follows:

IF User-Function = “Thesis Student" **AND** User-University = “HAW Hamburg"

THEN Get access to Master project room.

The above rule gives all users of organization “HAW Hamburg” with the function “Thesis Student” Execute access to the Master project room. Thus, a rule makes it possible to give one or more access rights to a whole group of users making rules quite a powerful and dynamic administration tool (17).

3.5.1 Role Vs Rule based Access Controls

The Role Based Access Control is already well established security concept. Its strong effectiveness comes from the fact of reflecting business roles without much technical stuff. It has very useful for auditing. On the other side there is a need of high administrative effort to assign the roles because of their static nature. Organization changes might cause the reassignment of some roles.

The Rule Based Access Control is relatively new security concept. The strongest aspect comes from the fact that it can assign the roles dynamically without much administrative work. The dynamic calculation of roles is based on user’s attributes. On the other side these dynamic assignments makes it difficult to audit the functionality as who is allowed to do what. As a result it is hard to maintain rules for long run because of its difficulty involved in foreseeing the impact of rule changes (17).

From the analysis of both role and rule based approaches it is clearly shows that the access control security mechanism of the document management system is well maintained using role based instead of rule based access control. Role based access control has clear organizational representation and better auditing capacity. It clearly shows the impact of each role and one can easily foresee its impact. This makes Role based access control as better choice for implementation of access control mechanism in the document management system.

3.6 Selection of Data Access Strategy

The selection of correct data access strategy helps to maintain the stability and performance of any system, which has a high data needs. The different techniques are as follows

Performing Database Operations Directly:

This model uses a data command object that includes an SQL statement or a reference to a stored procedure. This model has more control over execution of SQL commands or stored procedures and their returned values. Despite of this advantage it has a huge draw back in terms of development time and maintainability of the code. These days software development is highly undergoing layered approach for modularity and extendibility and better maintainability. One needs a better approach for data access than directly writing commands. Directly writing commands makes the data access code to scatter all over the application code. This makes impossible to maintain code successfully.

Creating Own Data Communication Layer:

This model involves writing own data communication layer's code using API's provided by different programming platform. Here one can have their own implementation and bring all the data persistence code to one place. The main drawback involves in the consumption of time for development of the code. It is better approach, if one chooses already existing frameworks rather than writing form the scratch.

Persistence Frameworks:

Persistence Frameworks addresses the major problem involved in the mismatch between the object- oriented programming sides to the relational database end. There are many persistence frameworks available in the market which simplifies the development process.

The different parameters for analyzing persistence frameworks are described as follows:

Object –to- Data: In this scenario mapping is started from the existing object model and maps those objects to the database tables. Finally uses persistence API to store and retrieve objects

Data-to-Object: In this scenario mapping is started from the existing database schema which is in XML or metadata representation, then generates the object model. Finally uses persistence API to store and retrieve objects.

Object and Data: In this case mapping is generated by using both existing object model and database schema, and use persistence API to store and retrieve objects.

	Hibernate	**Cayenne**	**MyBatis**
Object-to-Data	Yes	Yes	No
Data-to-Object	Yes	No	Yes
Object and Data	Yes	Yes	No
MySQL[11]	Yes	Yes	Yes(custom written)
PostgreSQL	Yes	No	Yes(custom written)
Oracle[12]	Yes	Yes	Yes(custom written)
SQL Server[13]	Yes	No	Yes(custom written)
Programming Platform	Java and .NET	Java	Java ,Ruby on Rails, .NET

Table 2: Comparison of Persistence Frameworks

There are number of best persistence frameworks available over open source communities. So comparison is only done among selected open source frame works in order to reduce the costs involved in development.

The table 2 clearly shows that Cayenne[14] persistence framework doesn't have many features and it mainly supports only Java platform. Most of the features in Hibernate[15] and MyBatis[16] are same but further analysis is needed in order to select better one.

[11] http://www.mysql.com/
[12] http://www.oracle.com/us/products/database/index.html
[13] http://www.microsoft.com/hk/sql/default.mspx

Simplicity: MyBatis is very simple to use and learning curve is so fast when compared with Hibernate.

Total ORM Solution: Hibernate is more traditional and regarded as complete ORM solution. The Hibernate maps objects directly to database tables, whereas MyBatis maps the objects to the results of SQL queries (18).

SQL dependency: Hibernate is the best option when one doesn't want to have much SQL queries. It generates efficient SQL at the runtime. However MyBatis gives complete control over queries (18).

Using across different databases: Both Hibernate and MyBatis can be used across different relational databases but the approach is quite different. Hibernate generates automatically the SQL code and one needs to write explicitly when using MyBatis. The Hibernate is better choice due to the automatic generation of SQL queries (18).

Performance: The Hibernate has caching facilities for gaining performance where as MyBatis gains performance by fine tuning the SQL queries

Using across different platforms: MyBatis supports Java[17], .NET (19) and Ruby on Rails[18] platforms whereas Hibernate supports both major platforms Java and .NET

The MyBatis is simple to use persistence framework and provides finer control over queries. Depending on different analyzed parameters and according to the requirements of the document management system, it is better to have Hibernate as a persistence framework.

[14] http://cayenne.apache.org/doc30/overview.html
[15] http://www.hibernate.org/
[16] http://www.mybatis.org/
[17] http://www.java.com/en/
[18] http://rubyonrails.org/

3.7 Selection of Website Framework

The Alpine Electronics Gmbh (7) had chosen .NET as their programming platform. The major Website Framework in .NET platform is DotNetNuke (20) and there is also framework named Cuyahoga (21) considered for analysis.

	Cuyahoga	**DotNetNuke**
Programming Language	C#	VB.NET
Costs	Free	Free
Web Server	IIS[19] , Apache	IIS
Operating System	Any	Windows
WYSIWYG Editor	Yes	Yes
Extra Features	Very Limited	Many
Search Engine	Yes	Yes
Database	PostgreSQL, MySQL, SQL Server	SQL Server
Mono Support	Yes	No

Table 3: Comparison of Website Frameworks

The table 3 is formulated in a way to reflect the main functionalities required for the document management system. In general DotNetNuke is well known and has better functionalities than Cuyahoga. As one don't need much ready-to-use features due to the development of the own system with specific goals. Cuyahoga uses both IIS and Apache (22) whereas DotNetNuke uses only IIS. This is very important if one needs to run on

[19] http://www.iis.net/

inexpensive server Apache. Another major concern is regarding operating system, the Cuyahoga can run on any operating system where as DotNetNuke can only run on windows environment. Independence on choosing operating system is always major advantage. Cuyahoga uses persistence framework as NHibernate (23) so that it can support any type of database whereas DotNetNuke only can support different versions of Microsoft's SQL server. Cuyahoga also can support open source .NET (19) development environment. It is quite clear that for the specific goal to develop document management system in very efficient and inexpensive way Cuyahoga is better choice.

3.8 Selection of Relational Database

The table 4 shows the comparison of different databases depending on the functionalities and costs associated with it. MS SQL server well suits to this environment because of the decision to use .NET as development platform. As NHibernate is carefully selected as persistence solution there is no need to have close integration with development environment. When one consider for open source database with more functionality it is clear that PostgreSQL (24) is better choice

Candidates	**Pros**	**Cons**
PostgreSQL	Open Source + large community Big functionality Good performance A GUI to view tables and make SQL queries on the fly	
MySQL	Open Source + large community Good performance	No GUI that is easy to use
MS SQL Server	Easy to integrate in IIS / ASP.NET	Costs Closed source solution

Table 4: Comparison of Different Databases

3.9 Selection of Web Server

The table 5 shows the comparison of several choices of web servers. Cassini has limited functionality which makes them out of competition. The main advantage of using MS SQL Server is due to integration with Microsoft's technologies. But MS SQL Server is not open source one needs to invest huge amount of money to obtain license, and it also limited to Windows[20] operating system. Apache (22) is well supported with both Windows and Linux operating systems; it has big open source community for support. Apart from that Apache has is use to setup and configure. It is clearly evident that Apache will be better suited as sever for the document management system.

Candidates	**Pros**	**Cons**
Apache	Open Source + large community Big functionality Good performance Highly extendable through modules Easy to setup & configure Can run on windows & Linux	
Cassini[21]	Very small footprint Integration in .NET	Limited to ASP.NET
MS IIS	Well integrated in the OS Works well with other MS technologies	Tedious to configure correctly Closed source product Only runs on windows

Table 5: Comparison of Web servers

3.10 Selection of Inversion of Control Framework

The table 6 shows analysis on .NET based Inversion of Control (IoC) frameworks, as the application is to be implemented on .NET programming platform. Sever parameters are taken for analysis of the suitable framework. Singleton is a software pattern, which means only one object is instantiated and only one time during the lifetime of the application. All the selected frameworks are supporting this pattern. Open generics injection is an ability to register open generic types and receive specific generic types on demand. This functionality is clearly not supported Spring.NET[22]. Auto mocking is an ability to leverage IoC framework to automatically mock dependencies of the tested component. This feature is available in Structure Map, but it can also be implementable

[20] http://www.microsoft.com/windows/
[21] http://blogs.msdn.com/b/dmitryr/archive/2008/10/03/cassini-for-framework-3-5.aspx
[22] http://www.springframework.net/

in future in Spring.NET and Castle (25). Although Structure Map[23] and Castle are having most of features finally Castle is selected basing into wide usage.

	Castle	Structure Map	Spring.NET
Singleton	Yes	Yes	Yes
Transient	Yes	Yes	Yes
Open generics Injection	Supported	Supported	Not supported
Auto-mocking	Not included but implementable	Included	Not included but implementable

Table 6: Comparison of IoC Frameworks

3.11 Overview of Selected Database components

The final phase is reached by identifying different components based on different parameters like cost, reliability, extendibility, integration etc.,

The figure 5 shows the overview of total database components and the explanation of each component is stated as follows:

***Incoming interface for Database*:** NHibernate (23) will be the incoming interface for the queries to the database. Hibernate mappings have to be defined properly so the persistency layer knows how to cascade the requests.

***Outgoing interface for Database*:** NHibernate will be also the outgoing interface for the results of the queries to the database. Hibernate mappings have to be defined properly so the persistency layer knows how to cascade the requests.

***PostgreSQL*:** PostgreSQL will be the database server for the system. It must not be necessary on the same machine than the web server and the web content. Communication is done via TCP/IP by using SQL requests.

The only client for PostgreSQL shall be NHibernate that provides an abstraction layer to the application.

[23] http://structuremap.github.com/structuremap/index.html

***Subversion*:** Subversion (12) will be used for all data that needs a history. This allows a rapid implementation of base lining based on tagging and ensures no data loss occurs, all data can be recovered, as subversion manages sets of changes and is incremental.

***NHibernate*:** NHibernate is used to abstract the database layer. The database is not directly visible to the web application and can be accessed in a more comfortable manner by using NHibernate mappings and API calls.

***Versioning*:** Versioning must be well handled there are several versions that must match:

- Version of the tables in the database
- Version of the modules

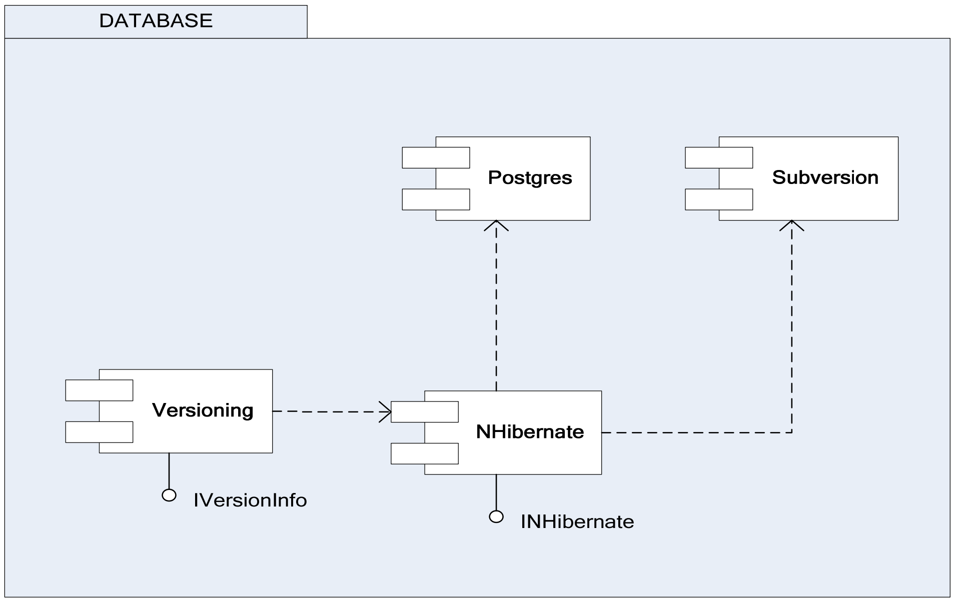


Figure 5: Overview of Database Components

3.12 Overview of Selected Web Server Components

Apache (22) is the front-end component seen by the web client. It makes use of several modules for extended functionality. By now, it will make use of Mod_ASP.NET to run an ASP.NET (26) Web application

***Incoming Interface*:** Clients connect to the system by using a browser. The protocol used for information exchange is HTTP, a standard protocol for the Internet that enables to keep the requirements on client side low. On the other side the Web application delivers the responses to the user requests.

***Outgoing Interface*:** The user requests are forwarded to the web application which in turn computes a response. The response is sent back to the client over HTTP by Apache.

***Apache*:** Apache is the container in which the modules are running and receives the connections coming over the network and handles them by delegating the functionality to the modules.

Apache is a complete Web server that makes it possible to quickly and easily deploy powerful Web sites and applications. When Apache receives a request, it examines the file-name extension of the requested file, determines which extension should handle the request, and then passes the request to it.

For ASP.NET (26), it will handle file name extensions such as .aspx, .ascx, .ashx, and .asmx. File extensions that are not mapped to any module are processed directly by Apache and returned to the user without any processing.

CHAPTER 4: RELATED TECHNOLOGIES

4.1 Subversion

Subversion is a version control system, which places the tree of files into a central repository to manage the files and directories at every instant of time. It remembers all the changes made to the files, directories. This helps to keep track of history of changes, getting older versions. The main reason for its popularity is due to the availability of repository across networks, which brings the convenience to be used by people on different computers. Collaboration is achieved by the modifying and managing same set of data from their own locations

4.1.1 The Repository

The repository is the central place for the storing of data. It stores information in the form of a typical hierarchy of files and directories. Clients will connect to the repository in order to perform read or write operation. Read operation brings the data from others to client, while write operation makes the data available to the other users by the client. Although this is function of file server system, the real magic comes from the capability to remember the changes.

In this typical system there is a possibility of overwriting the data by two concurrent users. Suppose two coworkers each wants to edit the same repository file at same time if first person make changes to the repository first, then there are a possibility the second person may accidentally overwrite with his own new version. Of course the first person’s changes won’t be lost because system remembers each change. But the changes made by the first person will not present in the second person’s newer version of the file, while the second person never knew the changes made by the first person to begin with. This kind of situation is avoided in subversion by using the *Copy-Modify-Merge Solution* (15).

4.1.2 The Copy-Modify-Merge Solution

In this solution every user’s clients contacts the project’s repository and creates a personal *working copy*, which is a local reflection of the repositories files and directories this process is known as *checking out*. Now users can work at the same time modifying their own local copies. Finally the local or private copies are merged together into a new final version. The process of saving the changes of local copies back to the repository is referred as *committing*. Ultimately human being is responsible to make it happen correctly. All this mechanism can be seen in the figure 6 (15).

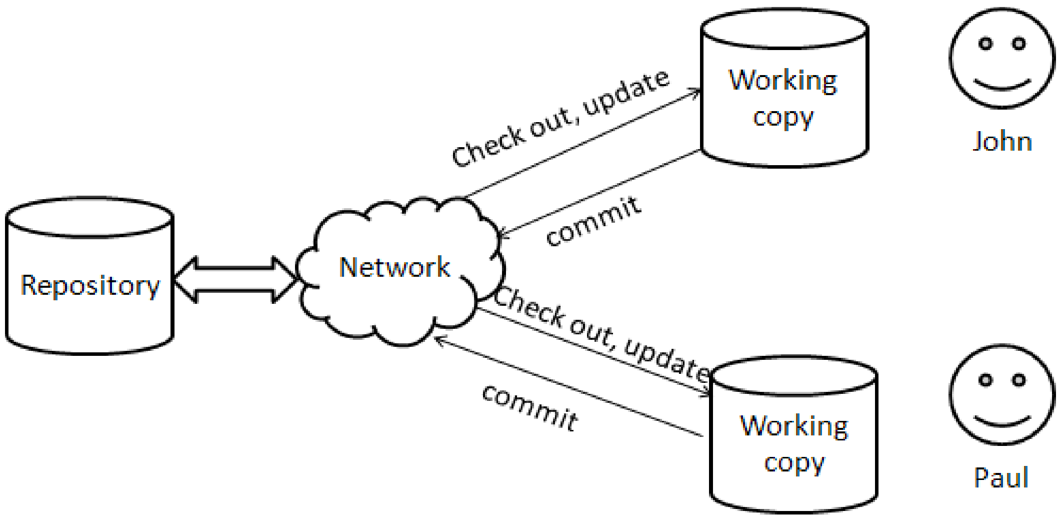


Figure 6: The Repository and Working Copies

4.1.3 Subversion Features

Subversion provides the following features (15):

- *Directory Versioning*: Subversion has virtual versioned file system that tracks whole directory trees over time.

- *True Version History*: The operations add, delete, copy, and rename both files and directories in subversion. And every newly added file begins with a fresh and clean history of its own.

- *Atomic commits*: Any operation will be completely occurs in the repository or not at all. This allows committing changes as logical chunks, and preventing problems from sending only part of a set of changes to the repository successfully.

- *Versioning of metadata*: The set of properties belonging to each file and directories are also versioned over time along with their contents.

- *Consistent data access*: In both text and binary files the binary differences are expressed using binary differencing algorithms. These files are equally compressed in the repository, and differences are transmitted in both directions across the network.

4.2 NHibernate

4.2.1 Object to Relational Mismatch

Essentially applications are object-oriented programs and relational databases store data in a relational form. There is no direct way to persist an object as a database row. This situation leads to the object-relational paradigm mismatch, which causes many hurdles in communication between object-oriented and relational environments. Some of the problems are as follows:

- *Identity and equality mismatch:* An Object-oriented programming language offers two distinct definitions for object equality and identity, whereas databases don't have corresponding clear distinctions.
- *Object inheritance mismatch:* Object can be inherited by another object; relational databases don't support the concept of inheritance. This results the necessity to have the own mechanism to translate class hierarchy to database schema
- *Problems of associations:* In an object-oriented approach, associations represent the relationships between objects. In relational databases, an association is represented by the foreign key column, with copies of key values in several tables. There is a noticeable difference between two representations (27).

4.2.2 Object Relational Mapping

It is a translation layer which can easily transform object into relational data and back again. This brings the overall solution to the object-relational mismatch. It isolates the business logic from any relational issues that may appear in the persistence layer. The main accomplishments using object relational mapping includes following benefits:

- *Modeling mismatch:* In order to gain development time the relational and object models must both have same entities. But in reality they may exist differently, the general solution includes redesigning the object model until it matches with relational model. This approach will consume time , by using object relational mapping one can overcome this situation
- *Productivity and maintainability:* Object relational mapping helps to concentrate more on business problems. It improves maintainability by reducing the lines of code and provides buffer between object and relational model.

- *Database independence:* This abstracts application from underlying SQL database and SQL dialect. Naturally it supports number of different databases and bringing portability to the application

- *Performance:* General claim is that hand-coded persistence is at least as fast as or faster than automated persistence. In order to be more precise about this fact there is a need to compare the effort being invested to both of these approaches. Many experiences have proven that a good object-relational mapping solution has minimum negative impact on performance. In some cases it even performs better than classic approaches. The architecture of Object relational mapping framework must be mature for performance optimizations (28).

4.2.3 NHibernate Architecture

NHibernate is the Object-relational mapping solution for the Microsoft.NET Platform. The following figure shows the main parts of NHibernate architecture:

As the figure 7 shows, the main components are NHibernate configurations files, mapping definitions, and persistent objects. The main component of NHibernate is its configuration. This configuration is always present by an XML, or a properties file includes the relevant information, such as database username, password, driver class, SQL dialect that NHibernate needs for connecting to a database, communicating with it, and performing persistence operations.

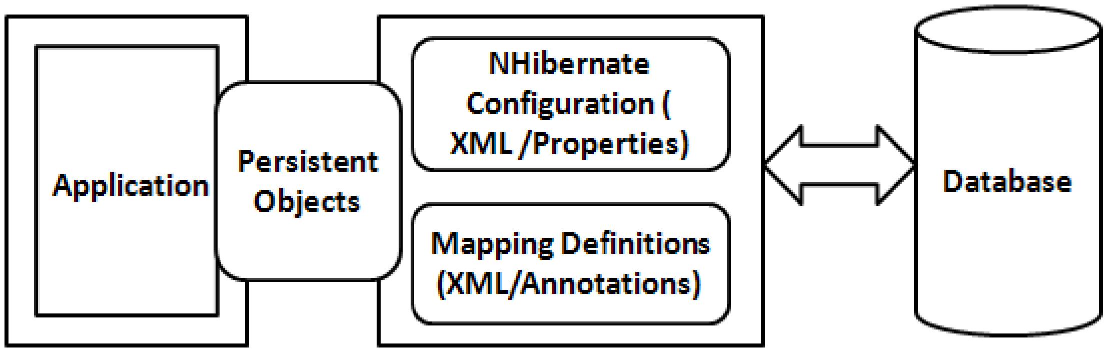

Application
Persistent Objects
NHibernate Configuration (XML /Properties)
Mapping Definitions (XML/Annotations)
Database

Figure 7: NHibernate Architecture
Source: (27)

The other part of NHibernate architecture consists of Persistent objects. These objects are persisted in the database. These entity objects and their classes follows the POCO (Plain Old C# Objects) rules.

4.3 Dependency Injection

4.3.1 Inversion of control (IoC) and Dependency Injection

Inversion of control in general context means reversal of responsibilities. In common use dependency injectors are commonly referred as IoC container. These terms are originated from the following principles (27).

- *Hollywood Principle*[24]*:* “don't call us, we'll call you”
- *Dependency Injector:* A framework that follows Hollywood principle
- *Dependency Injection:* The range of concerns with designing applications built on these principles

Inversion of control technology does the management of object dependencies by pushing dependencies into objects at runtime, instead of letting the objects pull their dependencies from their environment.

A software application consists of interfaces and classes. These together form application components. To perform functionality they will interact to provide requested services. These objects are dependent on each other, and object is called as *dependent* if it uses other object to perform its action. All other objects used by this object are termed as *dependencies*. The above explained dependency relationship can be depicted in the following figure 8.

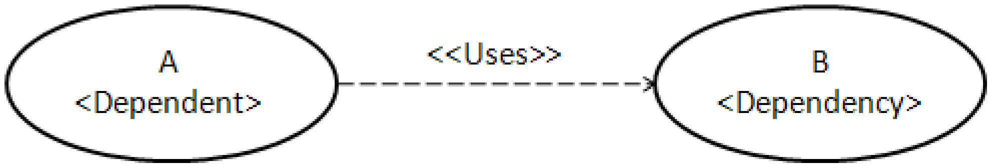


Figure 8: Dependency Relationships
Source: (27)

[24] http://en.wikipedia.org/wiki/Hollywood_Principle

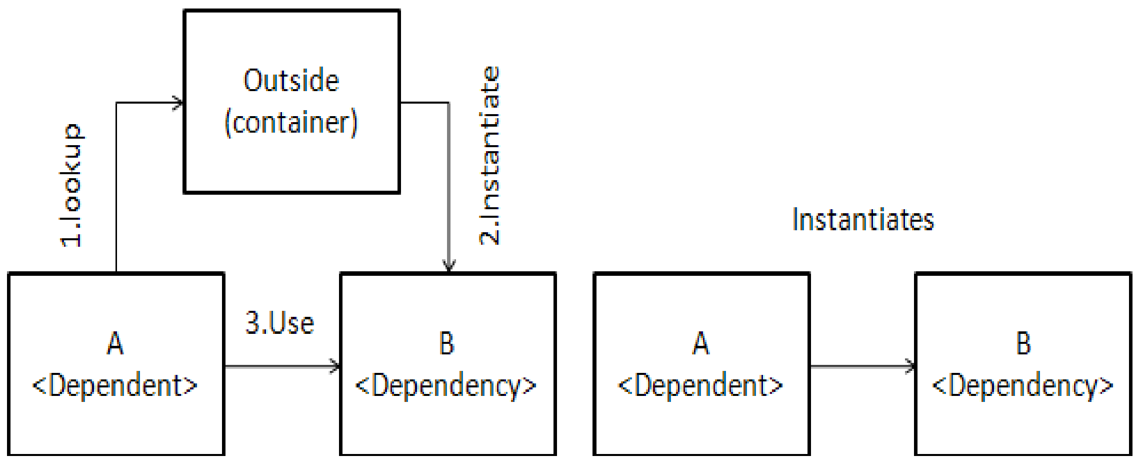


Figure 9: Non IoC Code Style
Source: (27)

The figure 9 shows the way how code works in non IoC style. The object calls its dependencies, when the object itself is responsible for providing dependencies from its environment. The object may carry this work by instantiating dependencies, or asking outside container object to provide it.

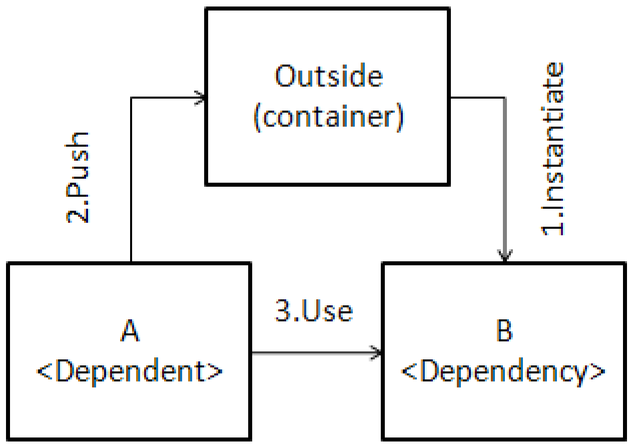


Figure 10: IoC Code Style
Source: (27)

In contrast figure 10 shows the IoC Code style, where object is free from providing its dependencies and outsourced to another object. This outside object will be responsible for instantiating the dependency object.

In this scenario, the object A has no longer instantiates the object B. Instead it is taken care by container object.

4.3.2 Advantages of Inversion of Control Container

- Application classes are designed very simply with minimum behaviors and required properties
- Application classes are self-documented, and documentation is always up-to-date
- Classes will not have their own configuration management, which in turn results the more manageable code
- The application leaves configuration management to framework
- There is increase in consistency since configuration management is done by the framework
- There is no configuration management code as framework handles this in every application (27)

CHAPTER 5: SYSTEM DESIGN

5.1 Overview of Complete System Architecture

The figure 11 shows an overview of the system architecture. Each of the big green rectangles represents a server machine. Of course all components may also run on one and the same machine if wanted.

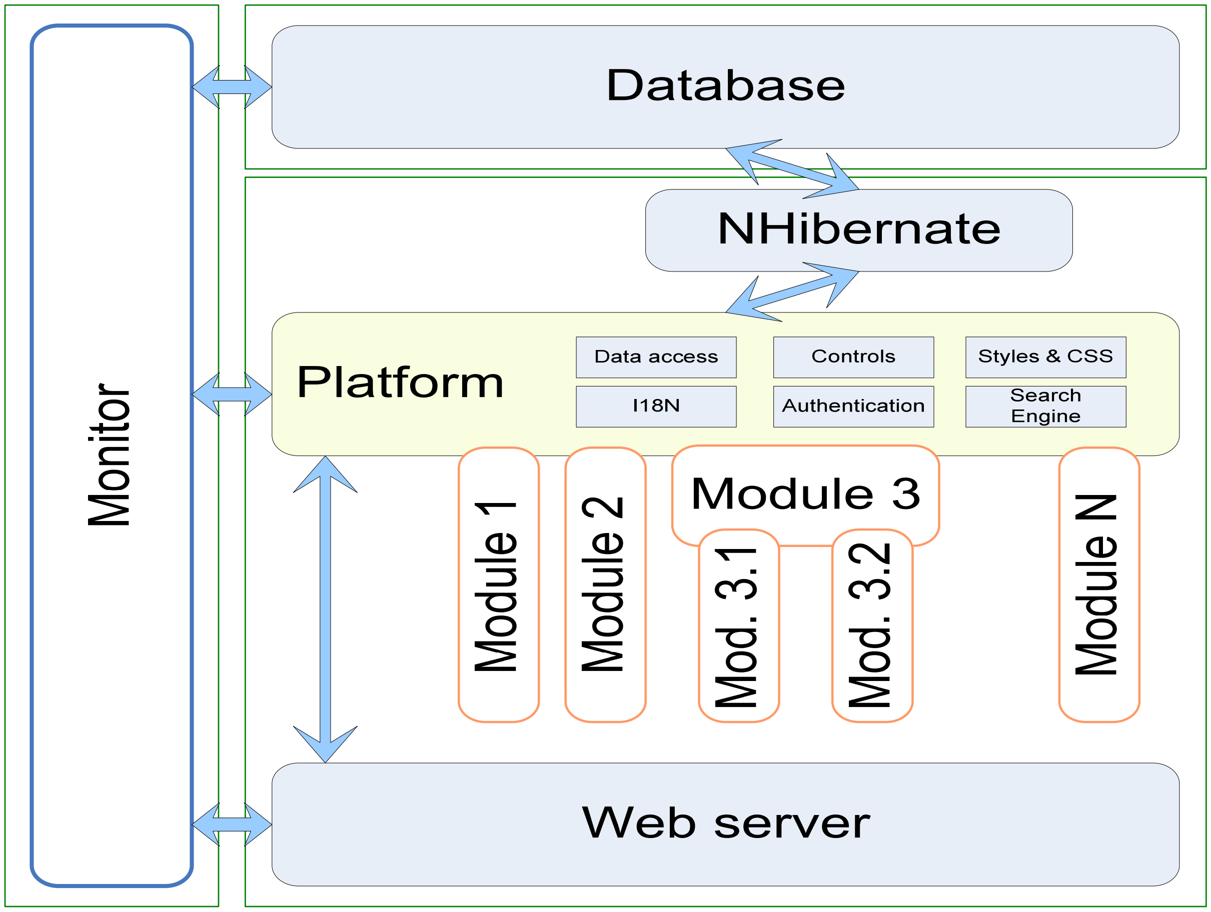


Figure 11: Overall Architecture Overview

Database Responsibility:

- The database layer provides data persistency. It is the central place where the data (all content) that does not belong to the framework gets stored. The communication layer (NHibernate based) also belongs to this component as it is closely related to it. However NHibernate (23) is used as an API by the client and therefore must not be installed on the same machine that is running the database server NHibernate is communicating with.

Database Interfaces:

- Data can be stored and retrieved by using queries (SQL-like) sent over a network connection.

Web-Server Responsibility:

- The web server provides the run-time environment for the web application and the connection point for external clients to interact with the system.

Web-Server Interfaces:

- A HTTP[25] connection point.
- Web server module API

Platform Responsibility:

- The platform provides a set of reusable components and a plug-in interface. This enables others to write modules that can be integrated and developed on their own with minimal invasion into the system.

Platform Interfaces:

- C# (29)Interface / Dynamic libraries

Module Responsibility:

- A module[26] provides a specific functionality. The module must use the platform to communicate with the database and the web-server (to enable central logging of activities). A module can have some private data associated with itself. In this case it is document management module.

Module Interfaces:

- C# Interface / Dynamic libraries, must use API provided by Platform

Monitor Responsibility:

- The monitor is responsible of setting up the system, allowing easy upgrades.
- Provides statistical information about the system like up-time, number of hits, users, …

[25] http://www.w3.org/Protocols/
[26] http://en.wikipedia.org/wiki/Modular_programming

- Provides ways to stress test the system

Monitor Interfaces:

- TCP/IP[27]
- OS services

5.1.1 In-depth Overview of Platform Component

The figure 12 shows the platform component context, where component exchanges data with the database and provides functionality to the modules or plug-ins. These also generate data that is transferred to the web-server.

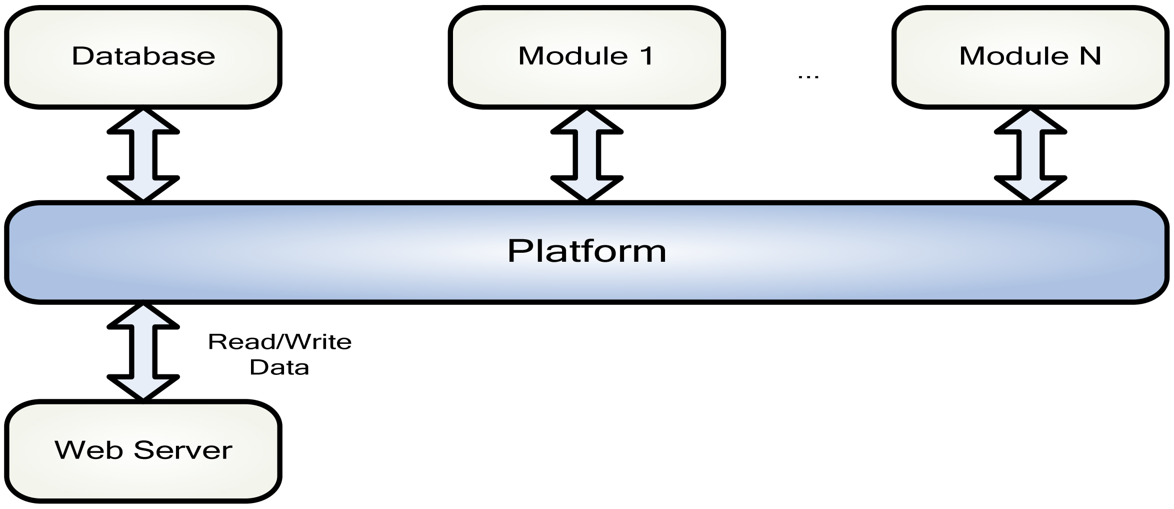


Figure 12: The Context of the Platform Component

5.1.2 Basic Component Platform Structure

In the figure 13 gives the structural overview of the platform component. The module web controls provides standard web controls like calendar, authentication dialog, search engine dialog and the core module provides core modules to display static HTML, to

[27] http://en.wikipedia.org/wiki/Internet_Protocol_Suite

download files, to manage users, where as module management (MOM) loads all plug-ins and determines which plug-in to call for a given URL. Also checks if the module version matches the version in the database and calls the update functionality accordingly.

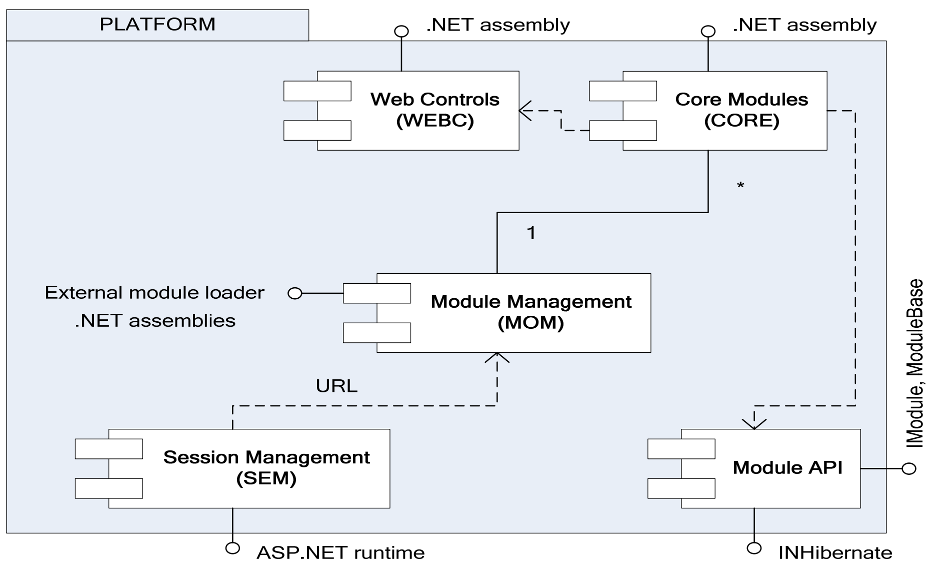


Figure 13: Structural overview of the platform component

The session management module provides session management, for each client stores the IP, connection start date/time, generates a session cookie and manages all the other session related data and module API can be used by all modules to implement their specific functionality.

5.1.3 Site Management by the Platform

The figure 14 shows the class relationships for the site management. The detailed description of objects in use is as follows:

- *Site Alias:* The Site Alias class enables mapping of an alternative URL to an existing site. Optionally one can specify a Node to where the alias should point.
- *Site:* The Site describes the website, the email of the webmaster and other site specific information that relates to the website.
- *Template:* It represents a template. This is not restricted to one physical file. It's possible to create multiple template objects based on the same template, user

control and style sheet. The template defines some placeholders for the content like sidebars, main content. These placeholders can be used to position sections on the page.

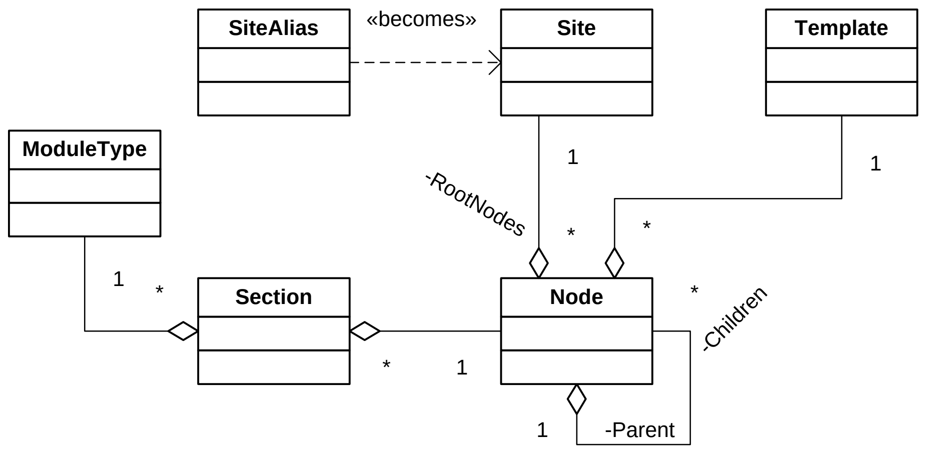


Figure 14: Class relationships for Site Management

- *Node:* The Node class represents a node in the page hierarchy of the site. A node is thus a page with a list of sections. A node has an associated template that defines the layout of the page and where to display the sections.

- *Section:* The section describes a section placeholder. It refers to an associated module which is responsible for displaying the content. The place where the section gets displayed is determined by a placeholder from the template in use for that page.

5.1.4 The System’s Directory Layout

The figure 15 shows the directory layout of the system, where by keeping the Monitor in the root directory, it will know where to find the components by itself and configure them for a first setup. This will ease the deployment and first installation of a complete system.

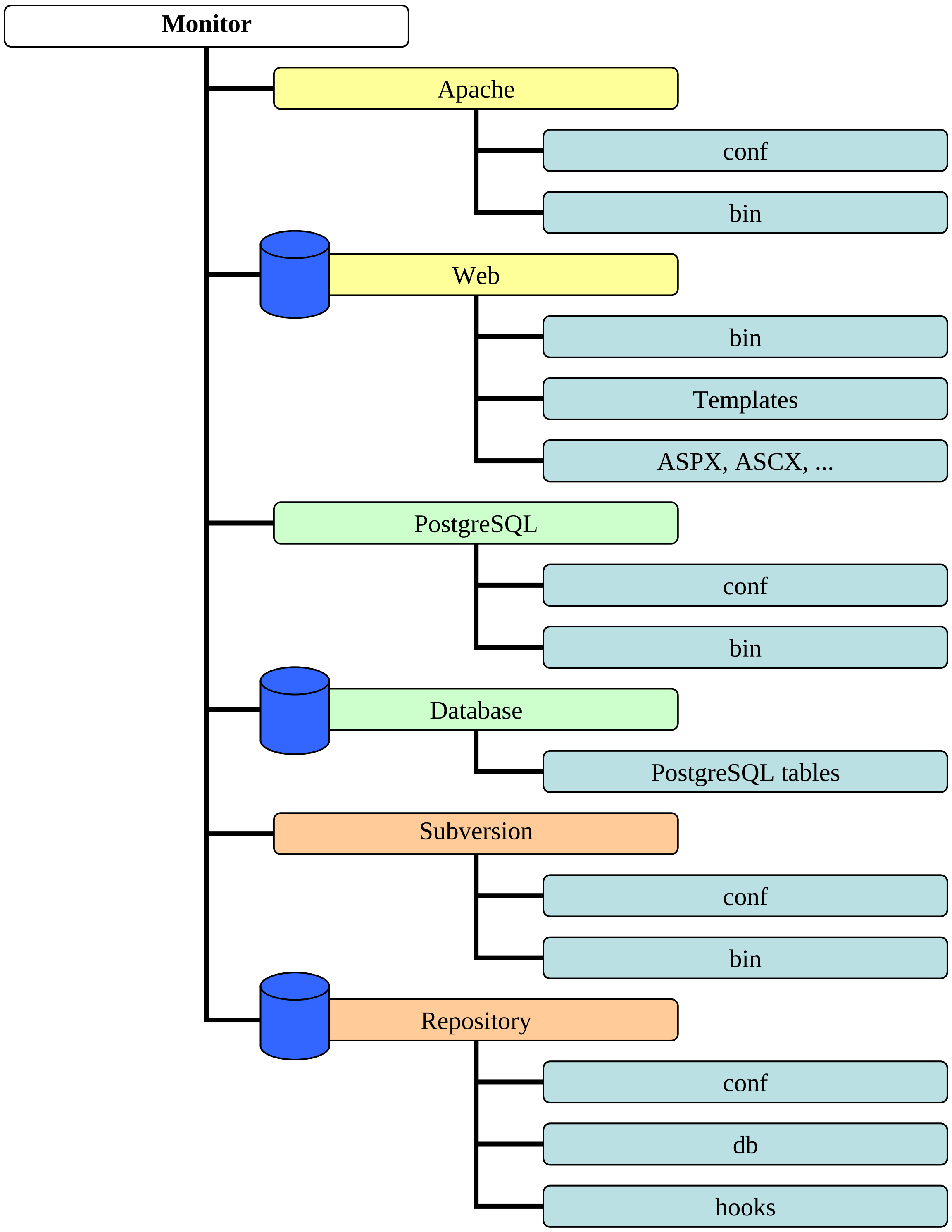


Figure 15: The System's Directory Layout

5.2 Functional Requirements Overview of Document Management System

The figure 16 captures the functional requirements of the system using Use Case diagram

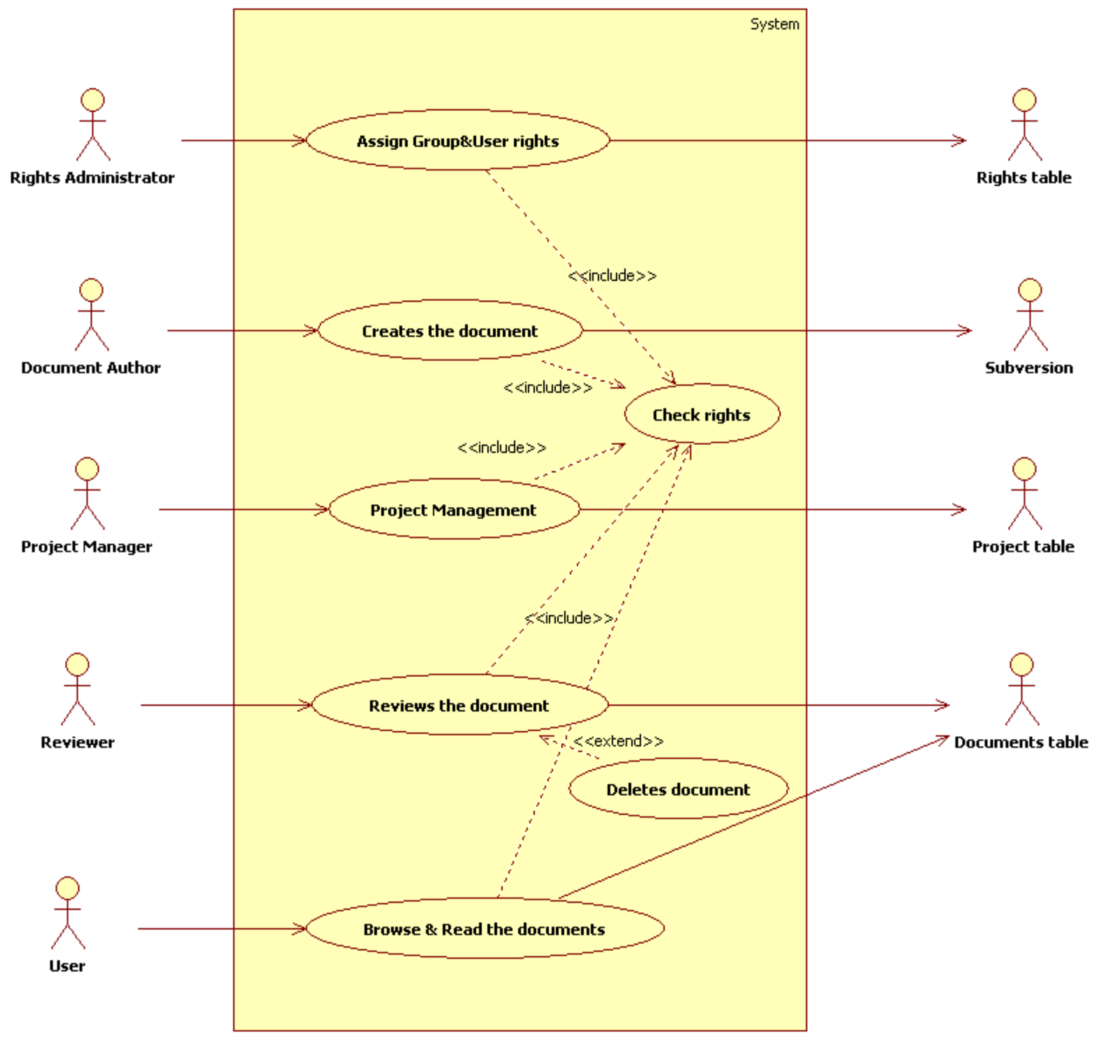


Figure 16: Document Management System Use Cases

- *Assign group & user rights:* The rights administrator can assign rights to the respective users. But it can be done only under his limits, which will be monitored by the check rights use case. The corresponding changes in rights must reflect immediately in the system. The newly updated rights are stored in the rights table of the database

- *Creates the document:* The document manager can creates the document and uploads it to the subversion repository .The rights of the document manager must be validated before this operation, which is clearly shown by using include dependency relation with check rights use case

- *Project Management:* The project manager administers the project and has rights to move, delete and create project. In some cases even assign rights to the project members. All these operations will be done based on his own rights, as shown by the check rights use case. The corresponding data is updated in the project table in the database

- *Review the document:* The reviewer sets the document properties like state. There is extending dependency on deletion of the document. All these operations has to be performed by checking the rights

- *Browse the document:* The User can browse through the list of documents and read or download them according to his rights.

- *Check rights:* This is the central point of authorization, where all the use cases must be validated according to their respective rights. This checking is done against the roles of the users.

5.3 Overview of the Document Management Module

The figure 17 provides the physical view of the system and shows the system's building blocks. The database transactions of the system are controlled by NHibernate component. So every module will communicate with NHibernate component to store or retrieve the data. In turn NHibernate communicates with the database component to perform the actual tasks. The document management and rights management module works together with the content management system (CMS) platform in order to produce required website functionality. The access control of the document management module is monitored by the rights management module. In order to save the actual documents in the Subversion document management module interacts with the Sharp SVN (30) component. In turn Sharp SVN performs the required operations against the Subversion. The in depth explanation of each component is as follows:

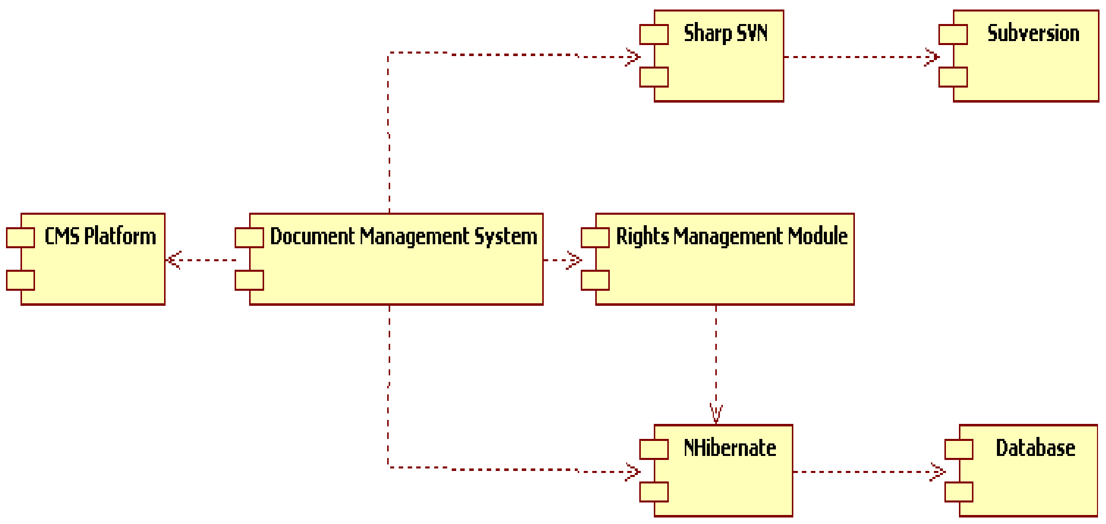


Figure 17: Physical overview of the system

5.3.1 Subversion

Subversion repository is used to store the data, which needs revisions. In the solution it is decided to use subversion as a place to store the actual document content.

The figure 18 shows the architecture of subversion. Subversion architecture consists of multiple layers. Each of these layers performs their specialized task with encapsulation and modularity (15).

- *File system*: This is the lowest layer and implements the versioned file system
- *Repository*: This layer implements many helper functions, which are built around the file system
- *mod_dav_svn*: It provide WebDAV[28] access to the repository
- *Repository access*: This layer manages the repository access for both local and remote access
- *Working copy*: This layer manages the local working copies which are local reflections of portions of the repository

[28] http://en.wikipedia.org/wiki/WebDAV

- *Client*: This layer uses the working copy library to provide common client tasks like authenticating the user and comparing versions

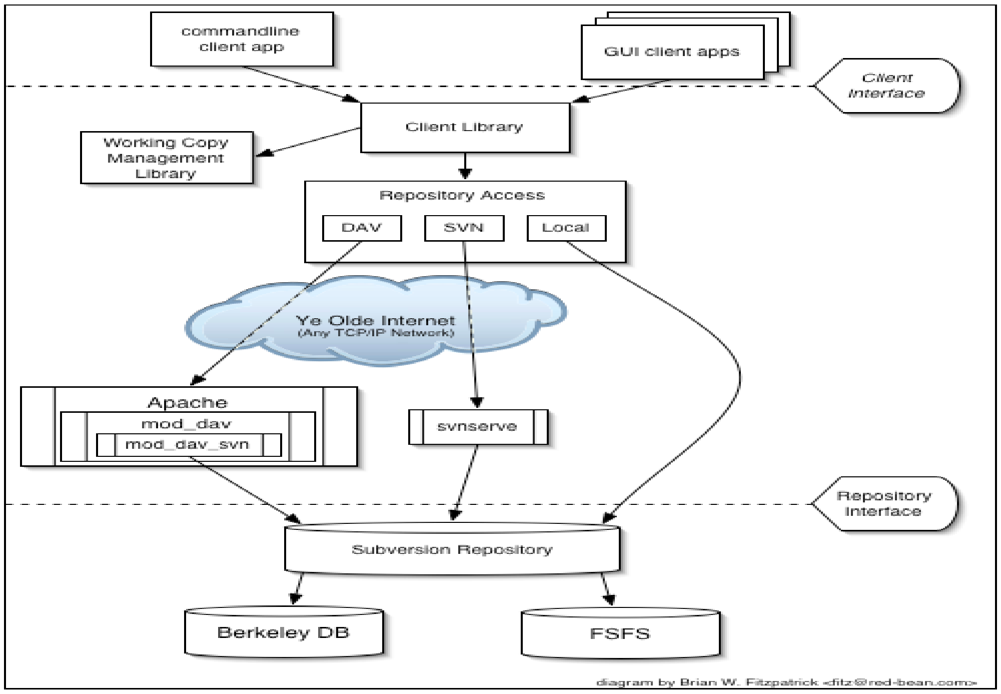


Figure 18: Architecture of Subversion
Source: (15)

5.3.2 Sharp SVN

Sharp SVN (30) is a simple to use API for binding with .NET based subversion applications. It also reduces the need to manually maintain the authentication call backs, apache portable runtime pools and other low-level things left dealing with while using other language bindings

Accessing a Subversion repository using Sharp SVN involves three steps:

- Open the repository
- Do the relevant operations (for example get the directory contents of the repository)
- Close the repository

In order to open a repository, one has to specify how they are going to access the repository. A Subversion repository can be accessed by any of the following three protocols:

- file:// To access a repository located on the local machine
- http:// or https:// To access a repository over the Web (using WebDAV protocol)
- svn:// A custom protocol proprietary to Subversion

Sharp SVN currently supports repository access using the file:// protocol

5.3.3 Database

Relational Database is used to store the metadata corresponding to the document management system, which gives the best way to index the data for search mechanism.

The process of design starts with identifying the purpose of database, there after organizing the required information and specifying information into tables, filling columns with corresponding items. The primary key specification helps to uniquely identify each row. One needs to find out how data in one table is related to other. These associations play a key role in data retrieving strategies. Normalization rules have to be applies in order to reduce redundant data.

The Entity Relationship model is shown in figure 19 gives the full overview of the data model.

- *cm_documents*: This is the principle table in database where each document is assigned an ID, title and document path is also stored here in order to retrieve the document. In this case the document path is the actual subversion path where the document is stored.
- *cm_user*: This table stores the required user's personal information like first name, last name, email, time zone. The most important information for authentication of the user like username, password is stored here and checked against each login.
- *cm_folders*: It stores the information regarding the folders. The corresponding folder of the document is obtained by having the association with *cm_documents* table using foreign key.

- *cm_dmrole*: This table has the definition of each role against its ID. This is the principal entity for defining the roles.

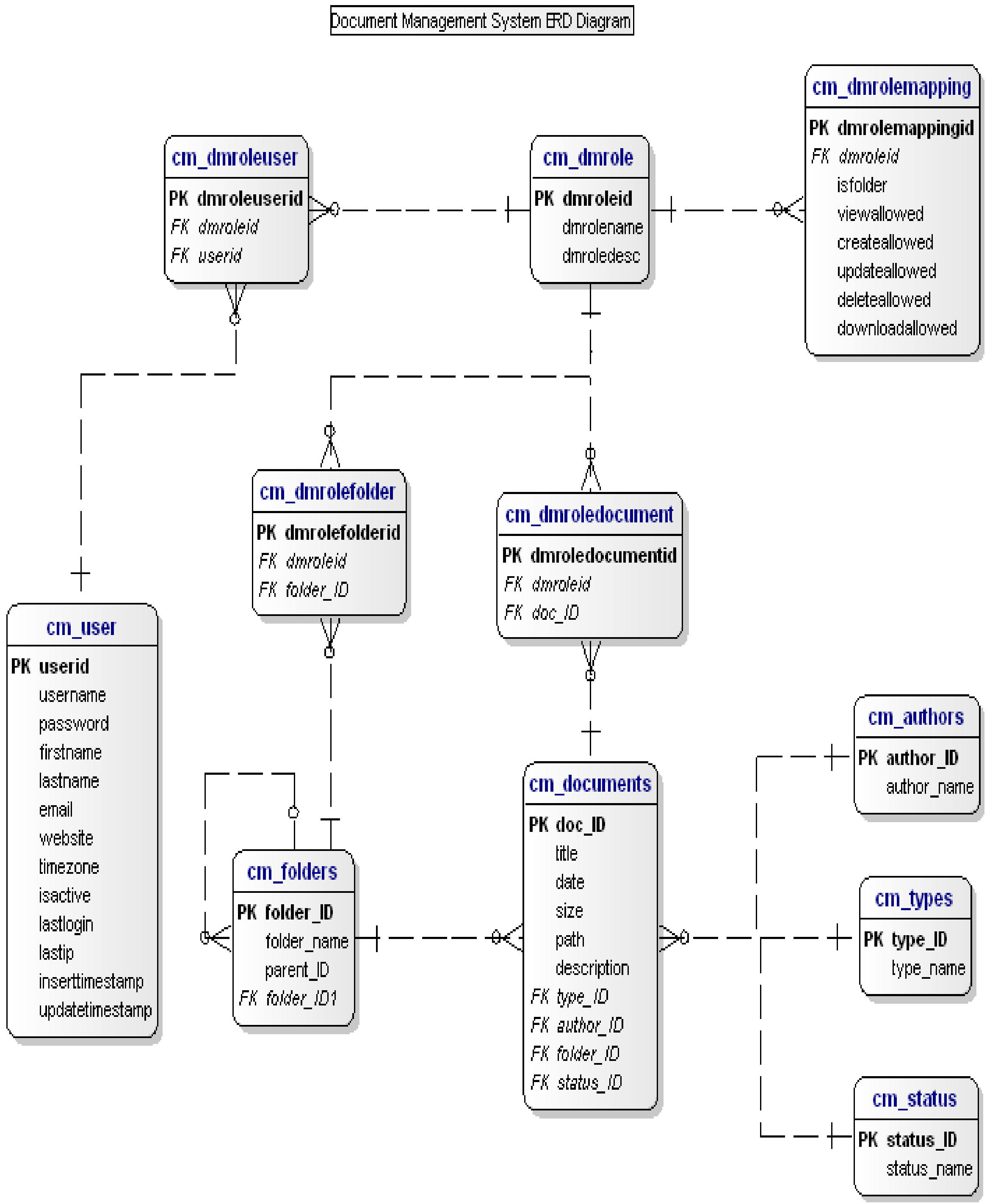


Figure 19: Entity Relationship Model

- *cm_dmrolemapping*: Here is the central place to link the roles with its corresponding operations. The bit fields like *viewallowed*, *createallowed*, *updateallowed and downloadallowed* will be assigned values either true or false according to the respective role. The association is established with the *cm_dmrole* table by means of foreign key.

- *cm_dmroleuser*: This table acts as intermediate linking table between *cm_user* and *cm_dmrole.* Here each user against its corresponding role is stored. The associations in either end are achieved by having one-to- many associations with the help of respective foreign keys.

- *cm_dmrolefolder*: This table acts as intermediate linking table between *cm_folders* and *cm_documents.* Each folder is associated with a corresponding role there by with the respective allowed operations. The one-to-many associations are established on either end.

- *cm_dmroledocument*: This table is the intermediate linking between *cm_dmrole* and *cm_documents* entities. The principle level of document security is preserved in this intermediary table. Each document is associated with an ID and similarly each role is associated with an ID, by interlinking both ID's one can assign the role to the document. The role permissions are stored in another table which is linked with role table. One-to-many association is held between tables as each role can be associated with many documents similarly in other way around many documents can have single role.

- *cm_authors*: Here the information regarding the author of the document is stored. Each author is held with an ID against corresponding name. There is a one-to-many relationship with *cm_documents* table. This relation helps to validate each document with its author.

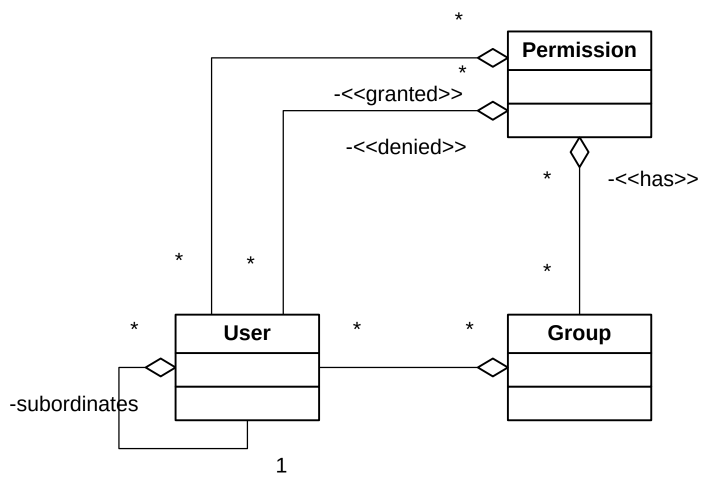
- *cm_types*: In general documents may have different formats; this information is stored in this table with a type ID against its corresponding format. It has a one-to-many relationship with *cm_documents* table in order to show the document with its corresponding format in the front end.

- *cm_status*: There is always a review process for each document, which is termed as status of the document. For example this information can be like accepted, rejected, updated etc. This table has one-to-many relationship with *cm_documents* table to link the document with its corresponding state.

5.3.4 Rights Management Module

Enterprise-scale organizations employ large numbers of internal users, with different access requirements spanning large numbers of systems, directories and applications. The dynamic nature of modern enterprises demands that organizations efficiently and securely provision and deactivate systems access to reflect rapidly changing user responsibilities.

This can be solved by using a hierarchical RBAC on one side that also allows special users to modify their subordinate's rights. This delegates the permission-updating task to several users and doesn't need a central administrator.



User:
Employee with his associated data (address, title, position, phone number, ...)
Each User might have several subordinates
Each employee has at most a chief or none if he is a CEO
Each user might be specifically assigned or denied a permission (override the group settings)

Group:
Aggregation of users
Each group may have several permissions

Figure 20: Class diagram for User, Group and Permission

The relation between user, permission and group can be clearly determined from the figure 20.

- *User*: Each individual communicating with system is a user with a user account. This user account identifies the individual when they log onto the server and personalizes the interface and data of the server according to the user permissions associated with the account through the role(s). The user permissions govern what the user can see and do.

- *Group*: Grouping users allows a role based access definition as one might set the permissions for a given group. Later one might decide which users belong to which group. Groups might contain other groups. Groups defined locally for a module may only be used within that module. Global groups and users are the responsibility of the Administrator for the "User Management" Module.

- *Permissions*: Permissions define what information a user can or cannot view, and identify the actions the user can perform. These actions include creating, reading, updating or deleting items. The administrator account is protected against their permissions being denied to stop accidental lockout from the software.

Access verification: The access rights to the object to be processed have to be verified. For example, during the display of a given node, following things are checked:

Access to the node must be granted (view permission for the page)

All sections (modules) where the user has view permission are displayed (all others sections and associated modules remain hidden and are so inaccessible)

- Edit, Administrate and other actions appear if the user has the corresponding rights for the section (module)

Effective permissions: The permissions calculated after evaluating the groups and the user's permissions. For a given permission, the user has the permission of the group (computed with an OR Boolean operator) if not otherwise directly set (overridden for his user id).

The workflow to determine if a specific user has a given permission is the shown in the figure 21. It first checks whether the user has the required permission then it cross checks if the user is in denied list or not. If the user is not in denied list then it checks against the granted list otherwise the access is denied for the user. After passing through the denied list then the system checks whether the user is in granted list from the positive reply, it goes to next step to check for the presence of user in corresponding group otherwise the access is denied. If the user also passes through the groups with positive reply then he is finally granted with the required permission

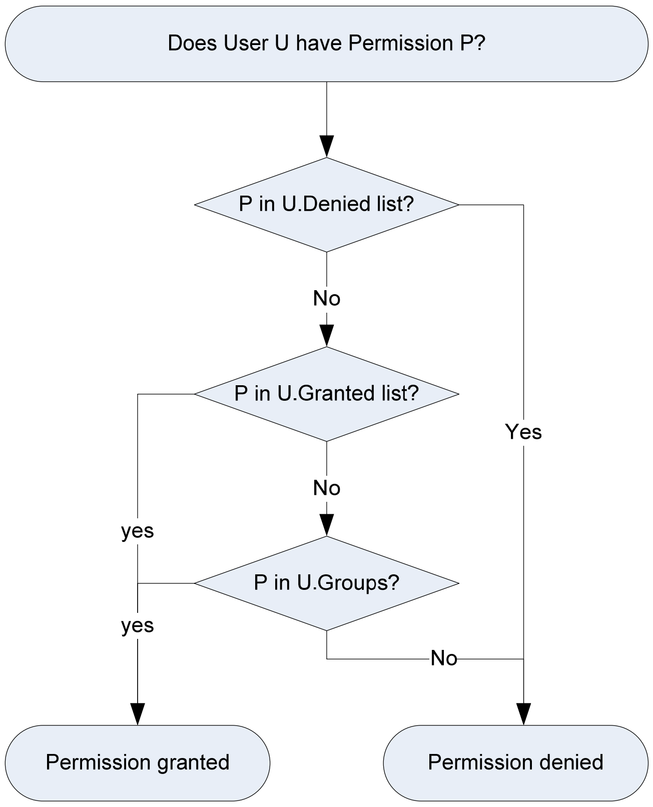


Figure 21: Workflow to determine if a user has a given permission

Authentication: To gain access to the server, users must log in. This process identifies and authenticates the person logging in against the user accounts. The user can be identified using a given authentication method (LDAP[29], custom local account). The method of authentication can be set for each user. The choice of authentication method will depend on the IT environment the users have, the organizations security policies, and user preferences.

The authentication sequence is clearly shown in the figure 22 .The interaction is visualized between client and server with respective interactions. After sending connect request to the server it checks, if the corresponding user is already authenticated. If the server sends reply by stating the authorization is needed then the client sends the required credentials to validate against the user. There after the server checks and creates the session ID for that particular session. There by the user is authenticated to send further

[29] http://en.wikipedia.org/wiki/LDAP

requests and gets the corresponding replies from the server. Finally after logging out request from the client the server will closes the session ID by making it inactive and deleted further.

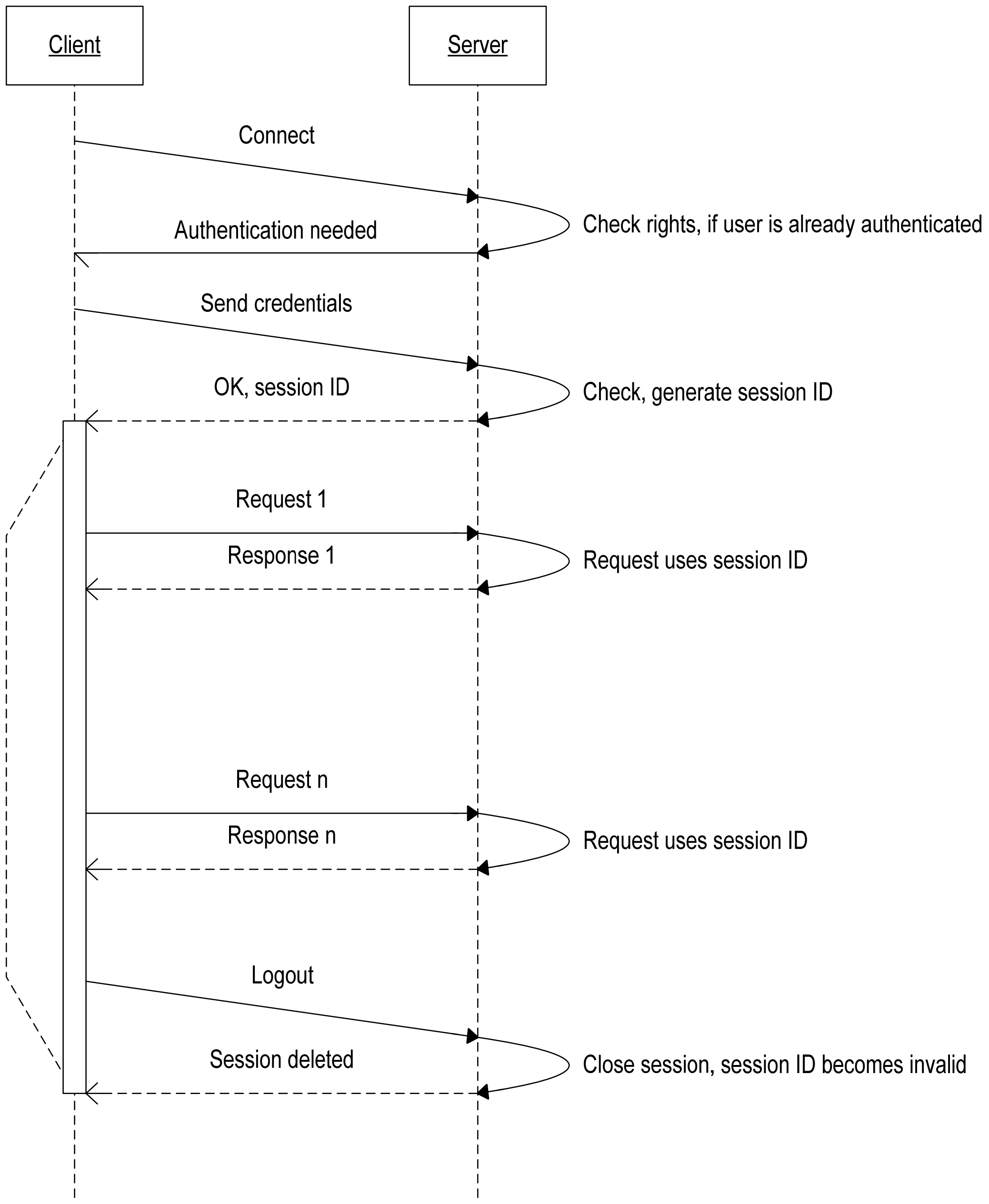


Figure 22: The Sequence diagram of a session

5.3.5 Document Management Module

The figure 23 shows the class diagram, which represents the basic building blocks of Document Management System in object-oriented way. It is more useful in illustrating relationships between classes. Aggregations and associations are all valuable in reflecting composition and connections respectively.

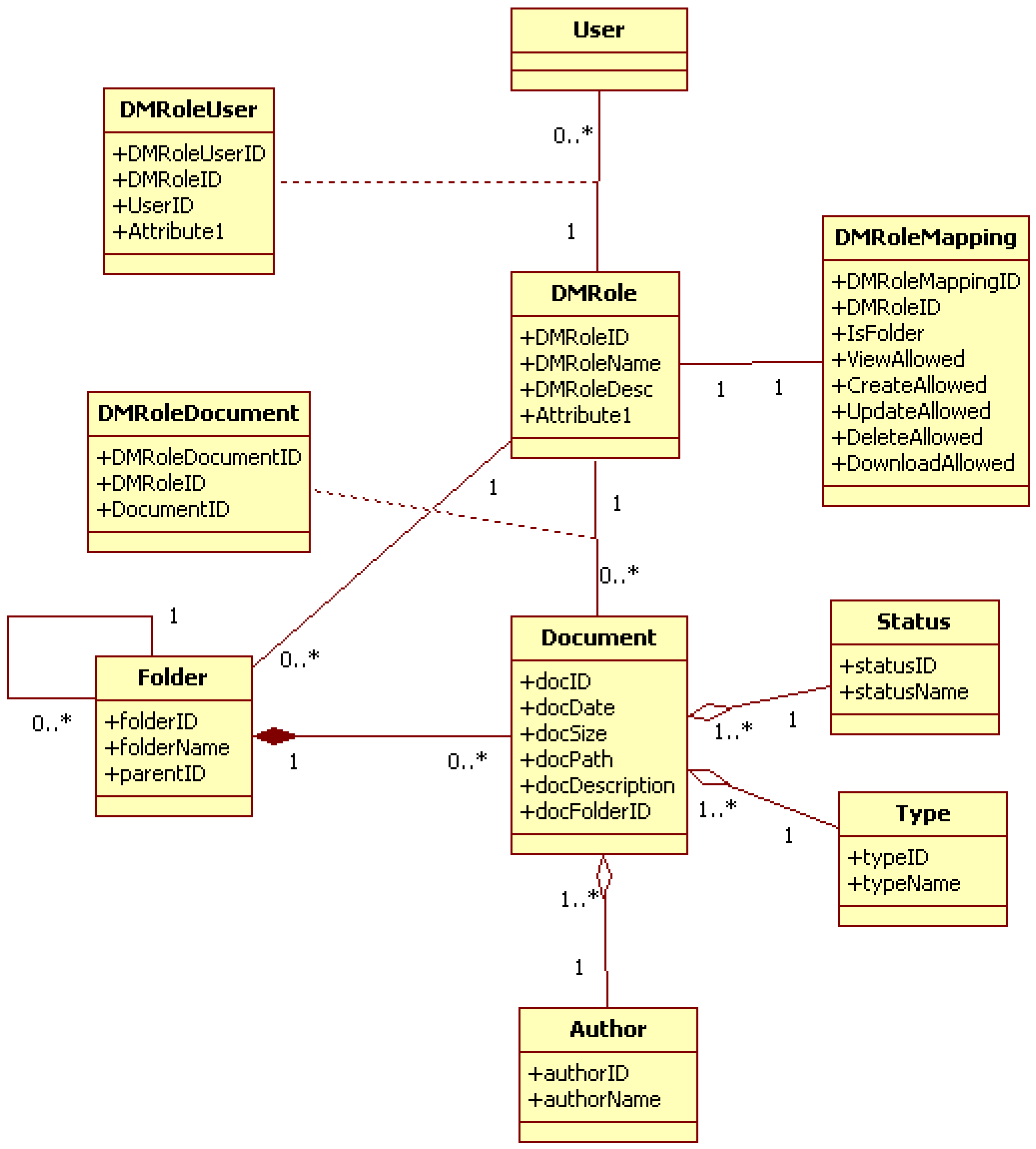


Figure 23: DMS Class Diagram

- *User*: This class contains the attributes regarding user’s information like username, password, email and methods for validating password, checking for permissions and further details.

- *DMRole:* This class has the defined roles of the system and their descriptions. It is the central place to the required roles.

- *DMRoleUser:* This class is an associative class, which relates the *User* and *DMRole* classes. It works as an intermediate linking class.

- *DMRoleMapping:* This class has definitions of the permissions for all actions related to *DMRole* class, like whether download, creation, updating and deletion of the document is allowed or not for particular role.

- *Document:* This has primary importance for assembling the total information required for the document object. This class consists of attributes like document title, creation date, document size, description and corresponding properties to provide encapsulation for the attributes. The document class is aggregation of *Status, Type* and *Author* classes.

- *Status:* After creation of the document it undergoes different levels of verifications and attains corresponding information named as status. This class consists of necessary attributes and functionality required for the storing and retrieving of status of the document. This class has one-to-many relationship with the *Document* class because many documents can have single status.

- *Type*: In general the system can handle different types of documents. All these types have to be clearly specified and stored for better understanding of the document. This class has many-to-one relationship with the *Document* class as many documents correspond to single type. In fact there is no need of having separate *Type* class. One can keep this into the document class itself, but the application is designed based on persistent entities. It will be easier if the classes reflects the tables in the database

- *Author:* The class consists of necessary attributes and methods for storing and retrieving the information regarding the author of the document. This class has one-to-many relation with the *Document* class because many documents can belong to single author and vice versa.

- *DMRoleDocument:* This is an Association Class for *DMRole* and *Document* classes. It provides necessary interlinking between these two classes.

- *Folder:* The Folder class consists of attributes like folder ID, name and Parent ID. Any folder can have sub folders for the required depth. The relationship has to be strongly preserved with respect to child and parent folders. This class has one-to–many relationship with Document class as many documents correspond to single folder. This class exhibits composite relation with *Document* class because the deletion of folder must include the deletion of its contained documents as well.

5.3.6 Workflow of the Document Management System

The figure 24 shows the basic workflow regarding download activity. When the user wants to download his required document, the system undergoes certain series of actions beginning from the start operation and then verifying the corresponding download rights in the database. If the user has corresponding rights then it gets the document stored path from the database otherwise it shows the message corresponds to the denial of the download activity and its possible reason. After having the download path, it issues download command to the subversion, where the physical documents are stored. Finally it brings the required document and saves it in the local specified folder.

This workflow clearly shows how the document management system, database and subversion work together to achieve required task. The system follows the similar flow for other activities like uploading and deletion of the document.

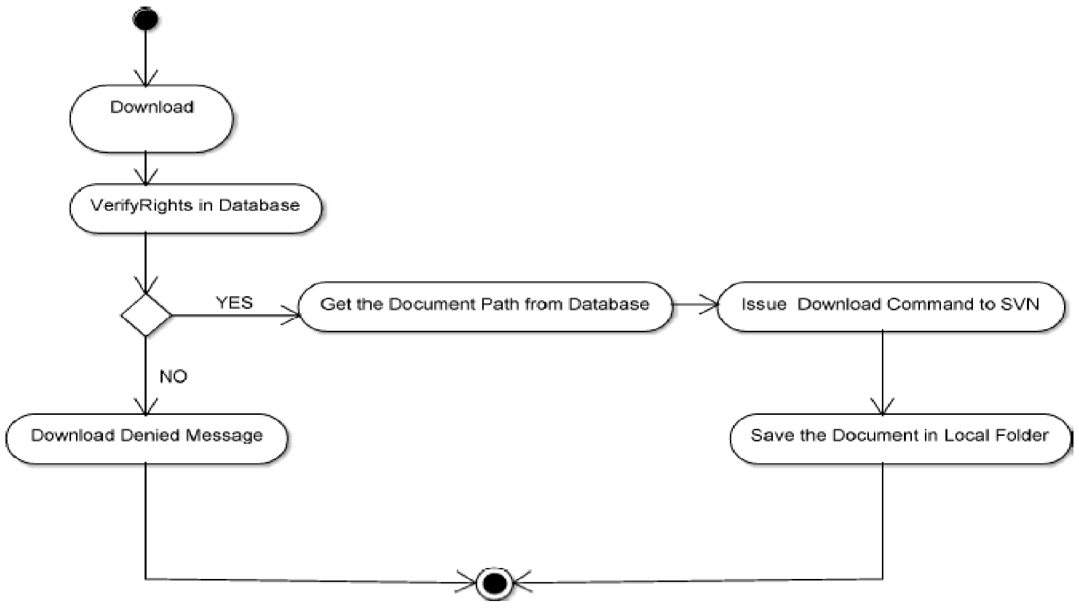

Download
VerifyRights in Database
YES
Get the Document Path from Database
Issue Download Command to SVN
NO
Download Denied Message
Save the Document in Local Folder

Figure 24: Download Activity Diagram

5.3.7 Document Management System Interactions

The interaction of various functionalities in the document management system is well viewed by the Sequence diagram shown in the figure 25. In this figure one can have different actors like user, subversion, database and document management system itself. The series of interactions and their corresponding replies are well shown here.

The parallel vertical lines shown in the sequence diagram gives the information regarding different processes or objects that live simultaneously and horizontal arrows gives the messages exchanged between them .This brings the specification of simple runtime scenarios in a well structured graphical manner

All the interactions shown in the figure 25 corresponds to the happy path in which one assumes every action is performed without any allegations.

- *Uploading the documents*: When a user tries to upload document, corresponding right is checked by system with the help of database .If the database returns true then the systems asks subversion to set the document and then system retrieves the version number from subversion and stores it in the database, thereafter the system shows success message to the user.

- *Download the documents*: The primary actor *"user"* initiates the download process and corresponding message is passed to the system. The system will initiates the verification request to the database and waits for its reply. When the positive reply is fed by the database to the system then it sends the *set document* request to the subversion, there by it downloads the corresponding document with respective success message.

- *Creation of folder*: The sequence of interaction corresponding to the creation of folder is initiated by the user request. Now the system will sends the request to the database in order to verify the respective rights against the corresponding user and waits for its reply. After receiving positive reply from the database, the system sends once again request to the database for the creation of the folder object. After successful creation of the folder object in the database it sends the success message to the user by confirming the creation of folder.

In the document management system database is used for creation of folder object and subversion to have actual document and the relationship between folder and its corresponding documents is well preserved in the database table with the help of document and folder ID's.

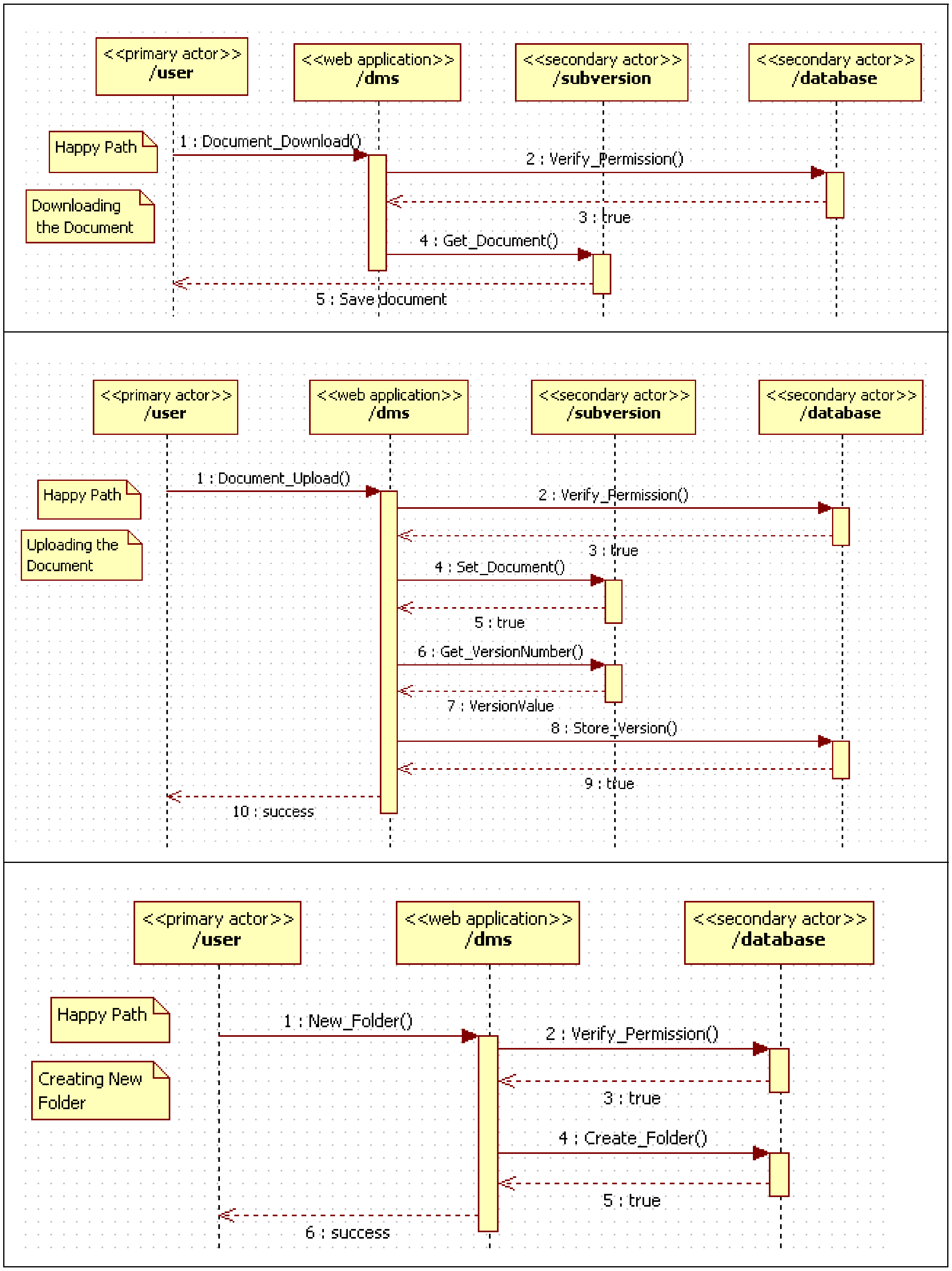
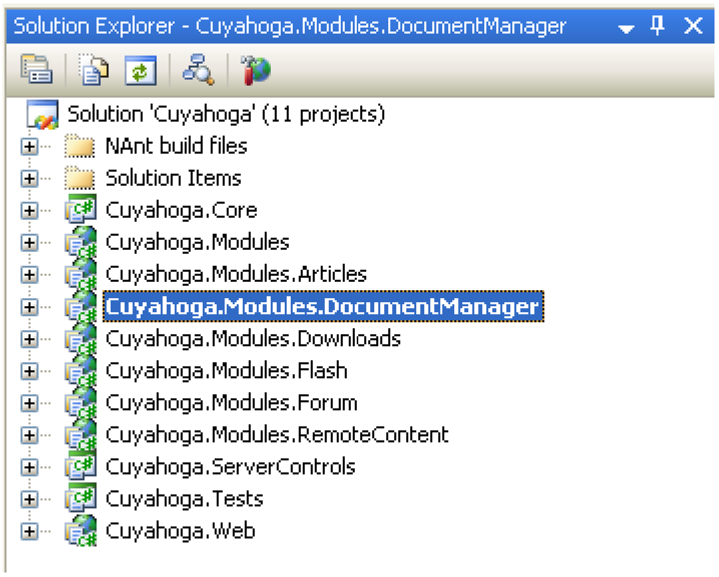


Figure 25: Sequence diagram for various functionalities in System

CHAPTER 6: IMPLEMENTATION

6.1 Modular Based Approach

The development of the whole system is modular based, which is part of Domain-driven design principles (31). A module is a group of features and functionality in the application. Modules help to organize large and complex domain logic into smaller and clearer units. In this application there is a core module, which contains the core functionality of the framework, web module to finally deploy on the server and the document management module which contains the implementation.



Solution 'Cuyahoga' (11 projects)
- NAnt build files
- Solution Items
- Cuyahoga.Core
- Cuyahoga.Modules
- Cuyahoga.Modules.Articles
- Cuyahoga.Modules.DocumentManager
- Cuyahoga.Modules.Downloads
- Cuyahoga.Modules.Flash
- Cuyahoga.Modules.Forum
- Cuyahoga.Modules.RemoteContent
- Cuyahoga.ServerControls
- Cuyahoga.Tests
- Cuyahoga.Web

Figure 26: Modular View of the System

In the figure 26, it is clear to see the whole solution of the project with its corresponding modules. The functionality corresponding to the implementation goes into the module named Document Manager. Module named Tests contains the test scripts corresponding to the project and the module *ServerControls* is for some common controls needed for the whole framework. Similarly module named Flash has the functionality to use flash technology in the system.

6.2 Layers Based Approach

The application will be easier to maintain, when it is structured in to different layers with separate concerns. Here the module mainly consists of three layers which are

- *Presentation*: This layer holds everything corresponding to the user interface of the application- the buttons, links and other controls that a user can click and interact while using the application
- *Business Logic*: This is the place for the rules of the application. It will consist of core functionality related to the document management module.
- *Data Access*: This layer is responsible for connecting to the data source and interacting with the data that is stored in that place. In this case the data source could be a database or subversion. In this layer the system is using NHibernate as a data persistence engine (31).

6.3 Model-View-Controller Pattern

This is a software architectural pattern, which isolates domain logic from the user interface and permits individual testing and development of each concern. The core phenomenon is described as follows

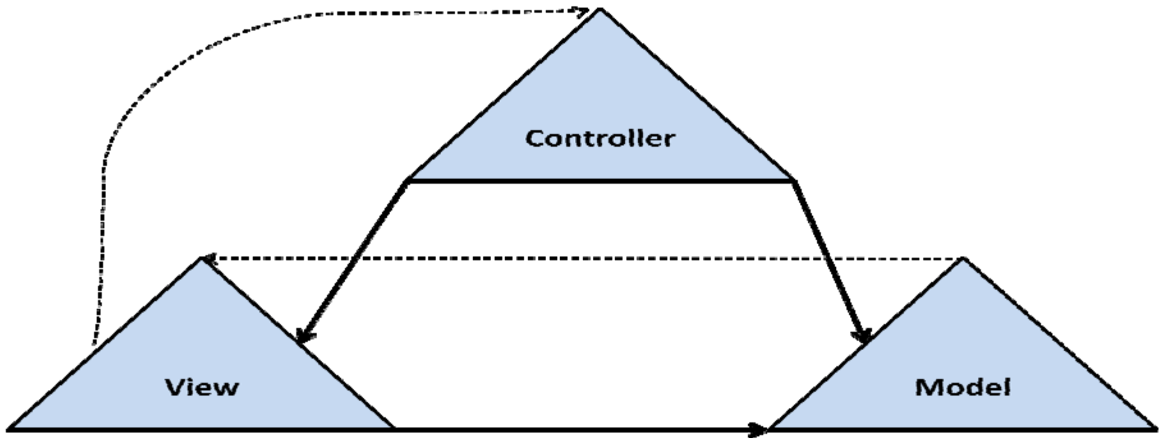


Figure 27: Model-View-Controller Pattern

The figure 27 clearly shows the three different parts names as Model, View and Controller (32).

- *Model*: The model is a direct representation of the business logic. It manages and notifies the information changes to the observers

- *View*: The view consists of .aspx and .cs files of the webpage. These files are responsible for defining the physical items that a user interacts in the web application. They are responsible for receiving the various events that a user raises while navigating through the site. The handled events must be passed immediately to the controller rather being handled in the view. It is the responsibility of presenter to handle each event. The controller will have the total control over the view.

- *Controller*: The controller is responsible for indirectly handling the events raised by the view and directly controlling what the view displays. It is also part of controller to communicate with the Model in order to access domain objects.

In the figure 28, it is clearly shown that the module is structured similar to the Model-View-Controller Pattern. The folder named *Domain* contains the business logic and contains the domain objects. The folder *Web* is similar to the view in the pattern, it consists of web controls, pages, style sheets and graphics related to the project. The file named *DocumentManagerModule.cs* is mostly similar to the controller. It contains or delegates functionality that doesn't belong in the *.ascx* controls or .aspx pages that are in module.

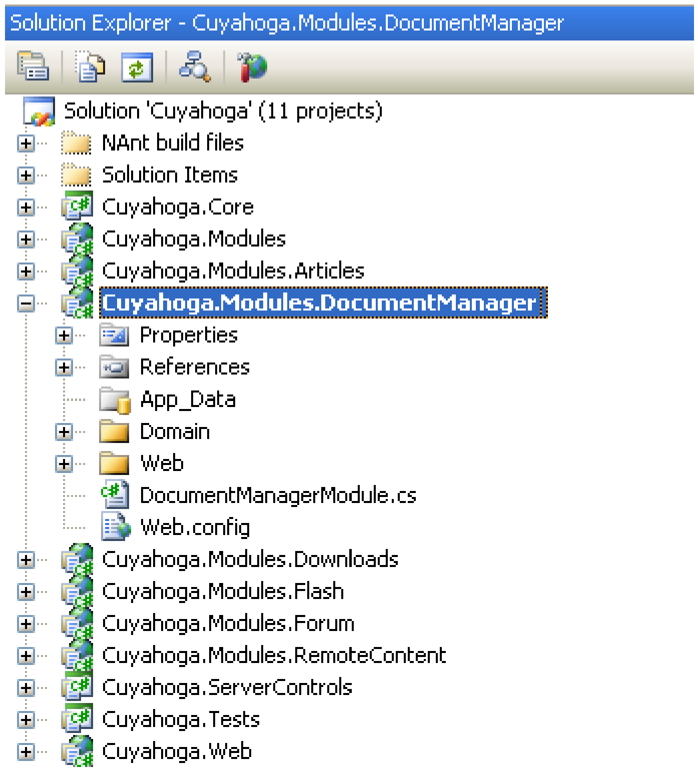
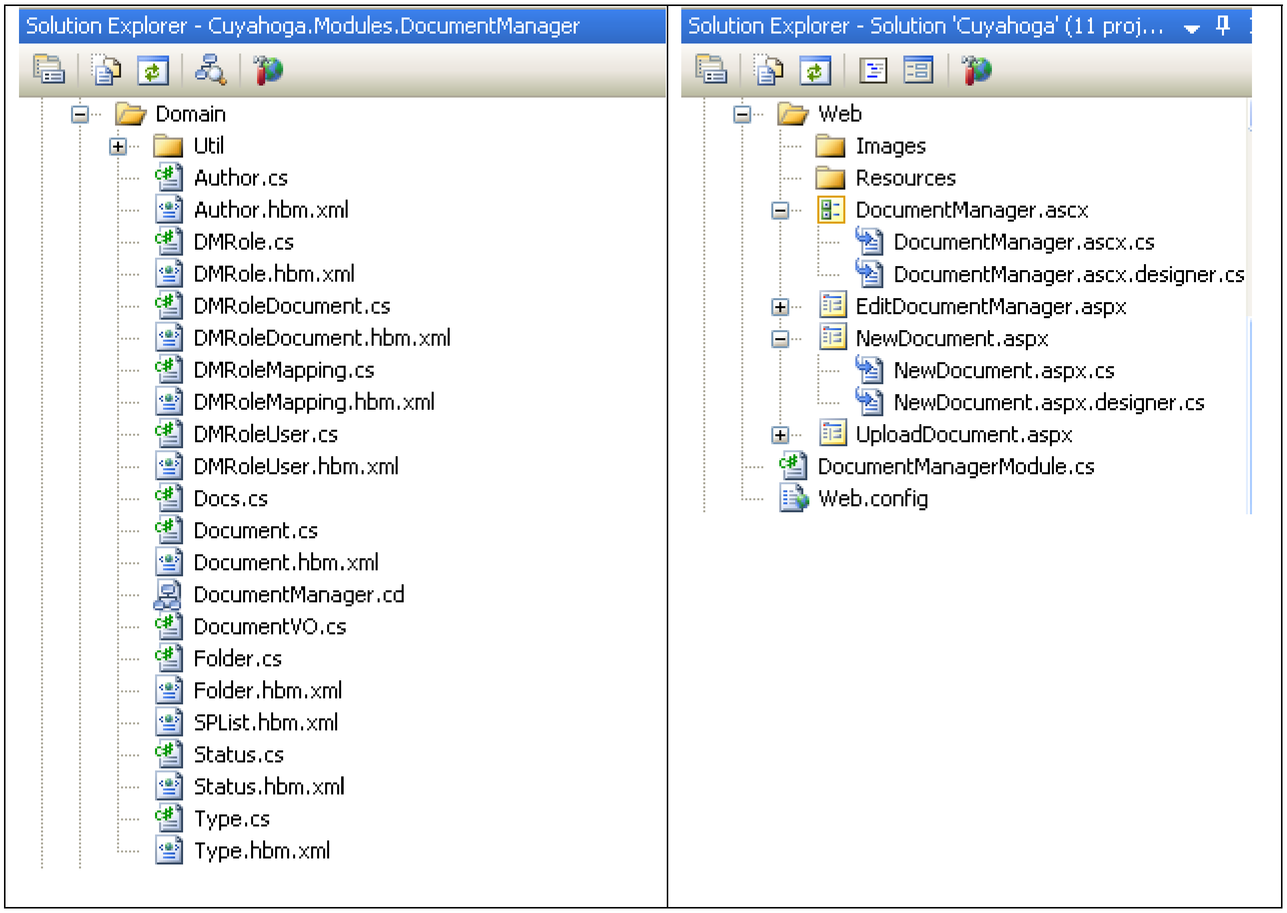


Figure 28: Inner view of the Document Management Module



```
Solution Explorer - Cuyahoga.Modules.DocumentManager
Domain
    Util
    Author.cs
    Author.hbm.xml
    DMRole.cs
    DMRole.hbm.xml
    DMRoleDocument.cs
    DMRoleDocument.hbm.xml
    DMRoleMapping.cs
    DMRoleMapping.hbm.xml
    DMRoleUser.cs
    DMRoleUser.hbm.xml
    Docs.cs
    Document.cs
    Document.hbm.xml
    DocumentManager.cd
    DocumentVO.cs
    Folder.cs
    Folder.hbm.xml
    SPList.hbm.xml
    Status.cs
    Status.hbm.xml
    Type.cs
    Type.hbm.xml
```

```
Solution Explorer - Solution 'Cuyahoga' (11 proj...
Web
    Images
    Resources
    DocumentManager.ascx
        DocumentManager.ascx.cs
        DocumentManager.ascx.designer.cs
    EditDocumentManager.aspx
    NewDocument.aspx
        NewDocument.aspx.cs
        NewDocument.aspx.designer.cs
    UploadDocument.aspx
DocumentManagerModule.cs
Web.config
```

Figure 29: Contents of Domain and Web Folders

The figure 29 has in depth view of Domain and Web folders. Here one can see all the files corresponding to the business logic in the Domain folder, which are having *.cs* extension. When one observes clearly, every domain class has also another file with similar name but with *.hbm.xml* extension. These XML files are called as NHibernate mapping files which plays an important role in storing and retrieving the domain objects. These files contain the information about how to persist the business classes.

The Web folder consists of two types of files with .aspx and .ascx extension. The files with .aspx extension are also known as web forms and contains the static (X) HTML markup, as well as markup defining server side Web Controls and User Controls where all the required static and dynamic code is placed. In the application the dynamic code is placed in another file with same name but having *.aspx.cs* extension. This style is called as code behind model where one gets clear separation between HTML tags and dynamic code. The other type of files is having *.ascx* extension, which are called as user control files. These user control files are containers to put markup and web server controls. User control files are much similarly organized as web forms but the key difference is user controls cannot run as a stand-alone files. Instead one must add them to existing ASP.NET (26) pages.

6.4 NHibernate Mapping

In the figure 30, it is clearly shown the basic structure of a mapping file *Document* class. This mapping file plays key role for retreiving and persisting the corresponding object in the database. In order to understand the functionality, it is necessary to know the metadata written in the XML file. The <*hibernate-mapping*> root element consists of all mapping definitions. The <*class*> is nested inside the root element, which defines the mapping of an entity class. The attribute *name* inside the <*class*> element indicates fully qualified name of the persistent class. The attribute *table* determines the name of the target database table in which the objects of this entity class are persisted. The attribute *lazy* enables or disables lazy fetching of associated objects.

The <*id*> element determines the object identifier and its corresponding primary key column in the table. This element also specifies how identifiers are generated for new instances of the class.

The <*property*> element maps a primitive property of the class, except for the identifier, to a particular column. The attribute *name* refers to the property name and *column* specifies the table column in which the property is persisted.

The next important concept to observe is mapping associations. The <*many-to-one*> element maps a many-to-one relationship. A relationship is called many-to-one when multiple instances of a class are associated with a single instance of another class. This relationship is usually carried out in the database by defining a foreign key in the many-class table which points into the primary key of the one-class table. The attribute name determines the name of the property inside the many-side class, representing the associated object (the one side). The class attribute represents the class name of the associated object. The default value is the property type determined by reflection. The attribute column specifies the foreign key column in the many-side table that points to the primary key of the one side.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
  namespace ="Cuyahoga.Modules.DocumentManager"
  assembly ="Cuyahoga.Modules.DocumentManager">

  <!-- Mappings for class 'Document' -->

  <class name="Document" table="cm_documents" lazy="true">

    <!-- Identity mapping -->
    <id name="docID">
      <column name="doc_ID"  />
      <generator class="native" />
    </id>

    <!-- Simple mappings -->

    <property name="docTitle" column ="title"  />
    <property name="docDate" column ="date" />
    <property name="docSize" column ="size" />
    <property name="docPath" column ="path" />
    <property name="docDescription" column ="description" />
    <property name="docFilderID" column ="folder_ID" />

    <!-- many-to-one mapping: Folders,Status,Types,Authors-->

    <many-to-one name="Folder" class="Folder" column="folder_ID"
cascade="save-update" />

    <many-to-one name="Status" class="Status" column="status_ID"
cascade="save-update" />

    <many-to-one name="Type" class="Type" column="type_ID"
cascade="save-update" />

    <many-to-one name="Author" class="Author" column="author_ID"
cascade="save-update" />

  </class>

</hibernate-mapping>
```

Figure 30: Sample NHibernate Mapping File

6.5 Application Configuration

The configuration setting corresponds to the whole system is described in the *web.config* file. It is worthwhile to know the few important elements in the configuration file. This is XML based text file. At runtime ASP.NET uses the configuration information provided here to compute the configuration setting for each individual resource.

The connection to the database is described in the configuration file as follows

```
<configuration>
 <properties>
            <!-- Database -->

            <!-- SQL Server settings -->
    <connectionString>server= \SQLEXPRESS;database=cuyahoga;Integrated
Security=True</connectionString>

<nhibernateDriver>NHibernate.Driver.SqlClientDriver</nhibernateDriver>

<nhibernateDialect>NHibernate.Dialect.MsSql2000Dialect</nhibernateDiale
ct>
 </properties>

</configuration>
```

The <*configuration*> element acts as a root element. The sub element <properties> describes the different setting for the connection of database. In this case the connection is with MSSQL Server. Due to use of NHibernate as a data persistence engine, one can literally connect to any database on their wish. The <*connectionString*> element is used to specify the type of database server , name and security mechanism. The <*nhibernateDriver*> element is used to declare the type of Sql driver.

The other important aspect to observe is the configurations settings of inversion of control container, which is *castle* framework in this case.

```
<configuration>
      <configSections>
            <section name="log4net"
type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
            <section name="castle"
type="Castle.Windsor.Configuration.AppDomain.CastleSectionHandler,
Castle.Windsor" />
      </configSections>

</configuration>
```

Configuration section handler declarations appear between <*configSections*> and </*configSections*> tags. Each declaration contained in a <*section*> tag specifies the name of a section that provides a specific set of configuration data and the name of the .NET Framework class that processes configuration data in that section. Here Castle Windsor is the Inversion of Control container and log4net is the tool used to output the log statements.

6.6 Overview of the total user interface

The figure 31 shows the total overview of the site. The total site is divided into three parts; the left plane is to display the hierarchical tree of the document folders. The upper right part is to view the list of documents in each respective folder. The documents will be loaded according to the User's selection of the corresponding folder. The below right plane consists of further properties of the selected document and the buttons to download, upload the document by the respective user.

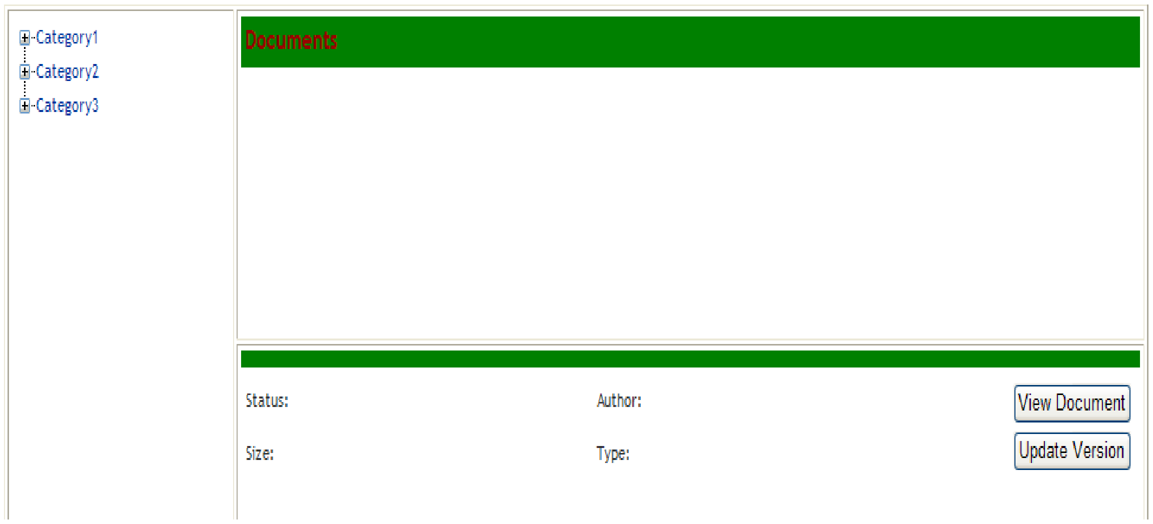


Figure 31: Total overview of the User Interface

6.7 Hierarchical view of the folders

The figure 32 shows the hierarchical view of the folders containing the documents. According to the requirements there must be parent-child hierarchical relationship among folders, which is clearly realized in the implementation. The implementation of the folders is created on the database rather than creating the actual folders and subfolders in the file system. This approach helps to have more convenience over folder data and maintaining its relationships. This also helps for making search across folders very easier. In general every folder has its ID and name corresponding to the folder and its parent folder ID. The Documents table constitutes the folder ID element against each document to materialize actual relationships between folder and its documents. This approach is better maintainable and productive and easier to implement when compared with the implementation of reflecting the real folders from the file system.

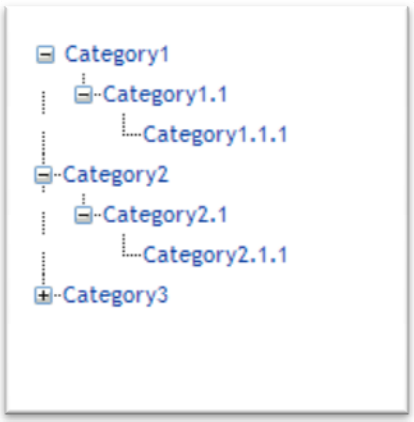
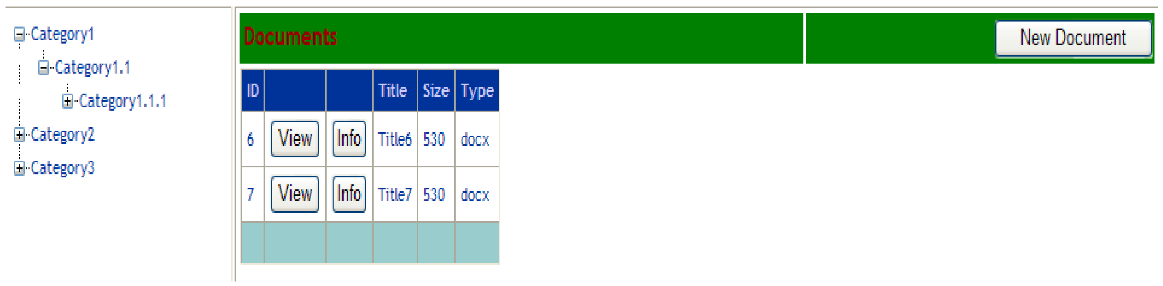


Figure 32: Folder Hierarchy

6.8 Documents Panel

The figure 33 shows the document panel, which populates the documents upon selection of the corresponding folder from the folders panel. The document panel is a structured as a grid view and contains various metadata like title, size and type of the documents. Apart from the properties it also consists of buttons against each document to perform particular actions like downloading the document or viewing further more information about the document.

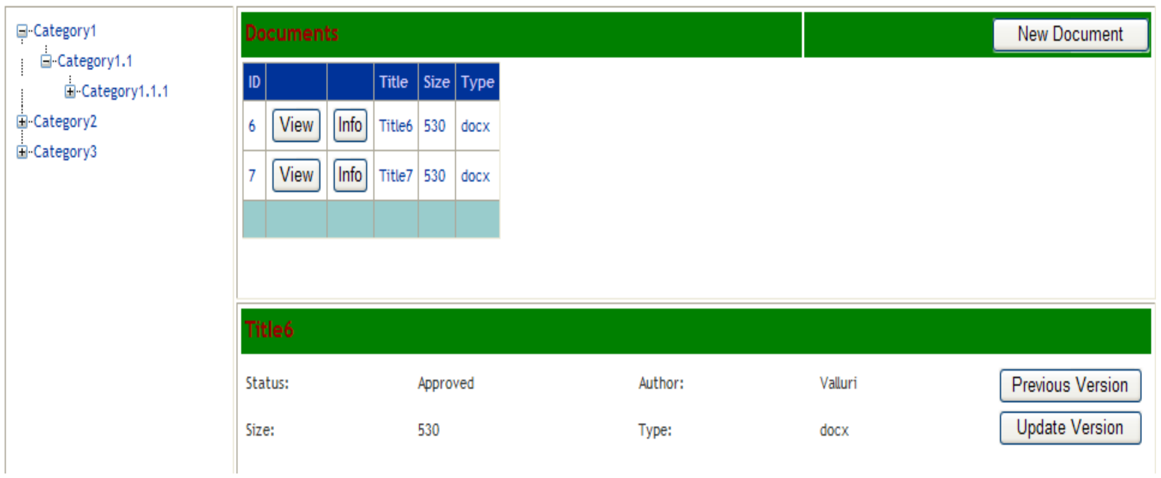
When the user clicks on View button then the system automatically downloads the document to the system’s folder.



ID			Title	Size	Type
6	View	Info	Title6	530	docx
7	View	Info	Title7	530	docx

Figure 33: Document Panel

In the figure 34 the lower panel is known as document detail-view panel. It brings further more information about the document, upon selecting the respective button located against the document. It shows the further information like status of the document and its corresponding author. Apart from this it also has the button named update version, which is useful to upload the new version of the current document in the system. It also has a button named previous version to download the previous version of the document.



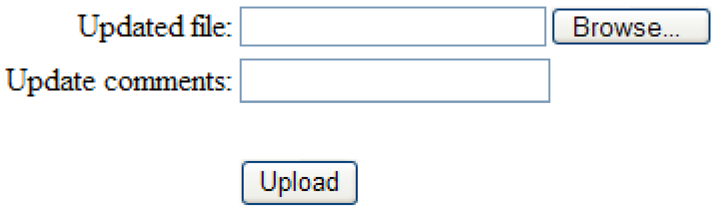
ID			Title	Size	Type
6	View	Info	Title6	530	docx
7	View	Info	Title7	530	docx

Title6

Status:	Approved	Author:	Valluri
Size:	530	Type:	docx

Figure 34: Document Detail-View

The figure 35 shows the page which will be redirected upon pressing the button named update version. This page constitutes two buttons for browsing the local document and uploading to the server. Upon successful upload the page will displays the success message

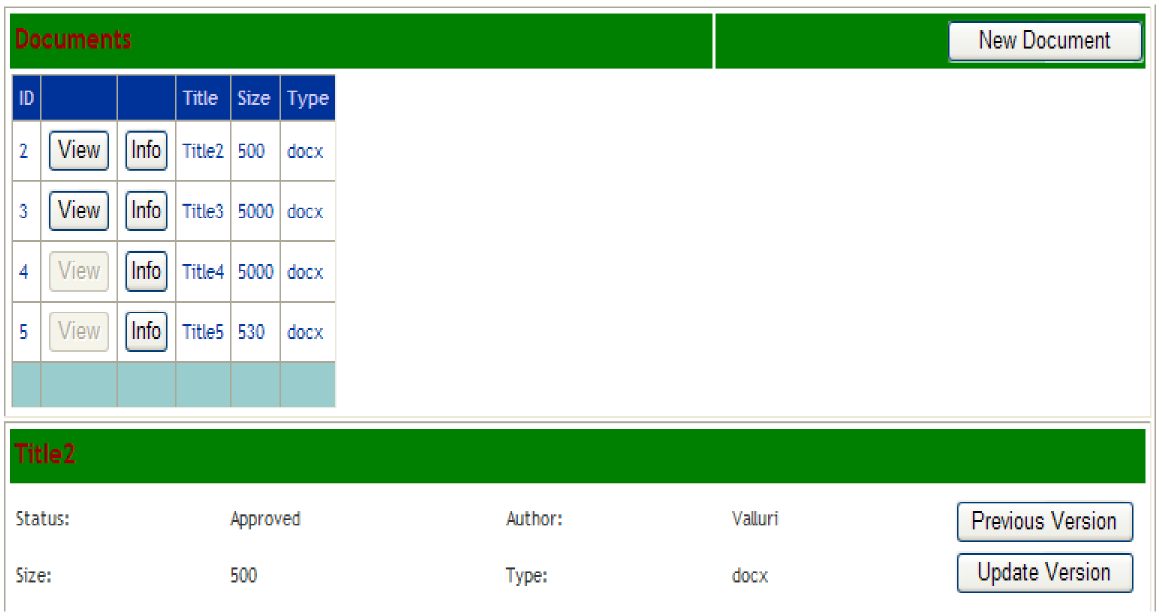


Updated file:	Browse...
Update comments:	

Figure 35: Updating the New Version

6.9 Checking Role Permissions

The document management system is developed in a way that the user interface view changes dynamically depending on the role of the users. Upon checking the permissions of the respective role user interface renders its contents according to the credentials. In this way one can clear filter unauthorized actions of the users.



Documents

ID			Title	Size	Type
2	View	Info	Title2	500	docx
3	View	Info	Title3	5000	docx
4	View	Info	Title4	5000	docx
5	View	Info	Title5	530	docx

Title2

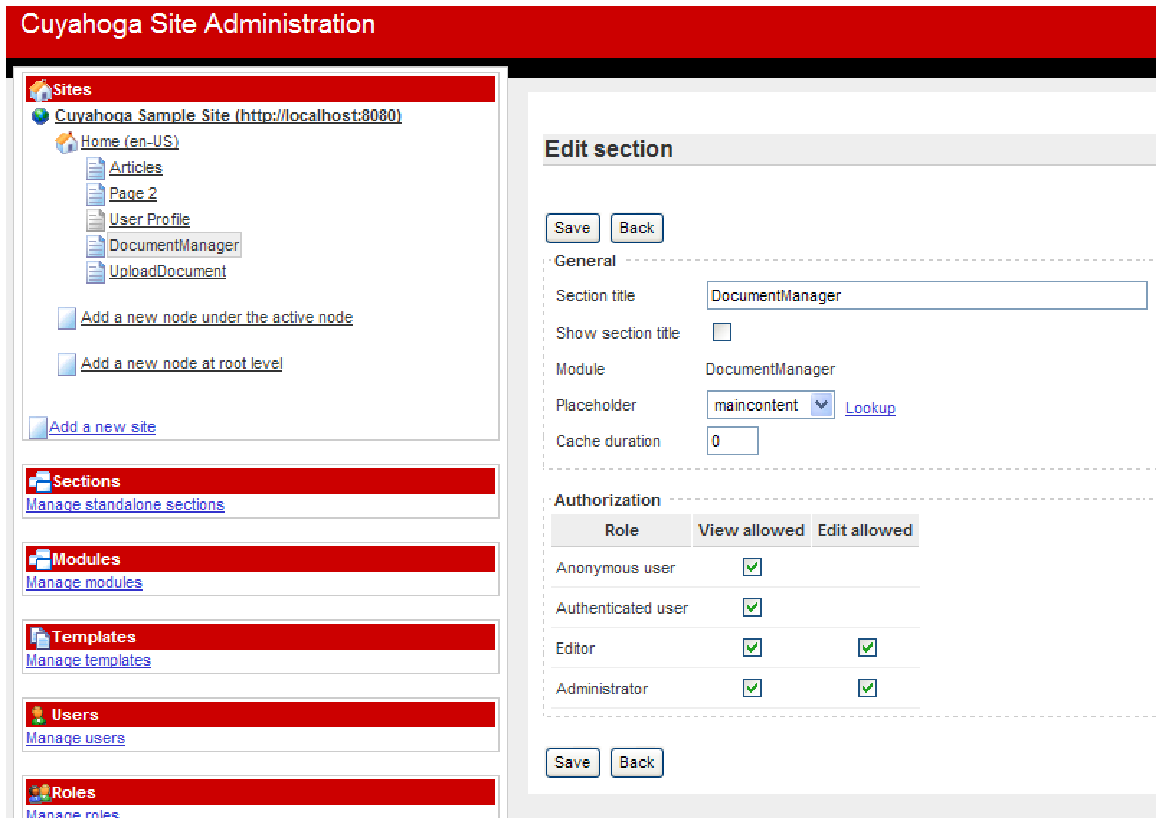
Status:	Approved	Author:	Valluri
Size:	500	Type:	docx

Figure 36: Grid view upon Checking Permissions

The figure 36 reflects the facts, when the user selects the particular folder to view its contents. The corresponding documents are loaded in the document panel. The figure 36 shows that grid of the document panel doesn't have particular buttons activated against some documents. This is from the fact that the system checks the credentials of the user and assigns the respective role. When the role doesn't have the permission to do particular action on the system, the corresponding functionalities are disabled from the user interface. In this way system is well behaved by justifying the roles importance. The new technique involved in developing this role mechanism is that, the roles can be individualized in the document level rather than conventional based folder level.

6.10 Site Administration

The document management system is implemented as a module in the content management website framework. It is worthwhile to know how to administer the module by using its user interface.



General

Section title	DocumentManager
Show section title	☐
Module	DocumentManager
Placeholder	maincontent
Cache duration	0

Authorization

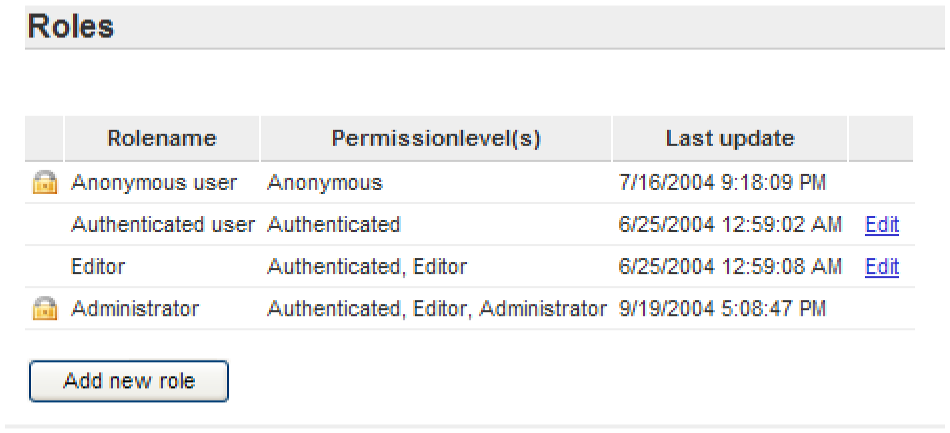
Role	View allowed	Edit allowed
Anonymous user	✓	
Authenticated user	✓	
Editor	✓	✓
Administrator	✓	✓

Figure 37: Website Administration

In the figure 37 it shows how to create initial website for each modules using the administration webpage. Every new module is considered as sections in the total content management system. At the time of site creation one can assign the role permissions to the corresponding section. The content of the newly created section can be positioned according to the different available placeholders.

6.11 Role Administration

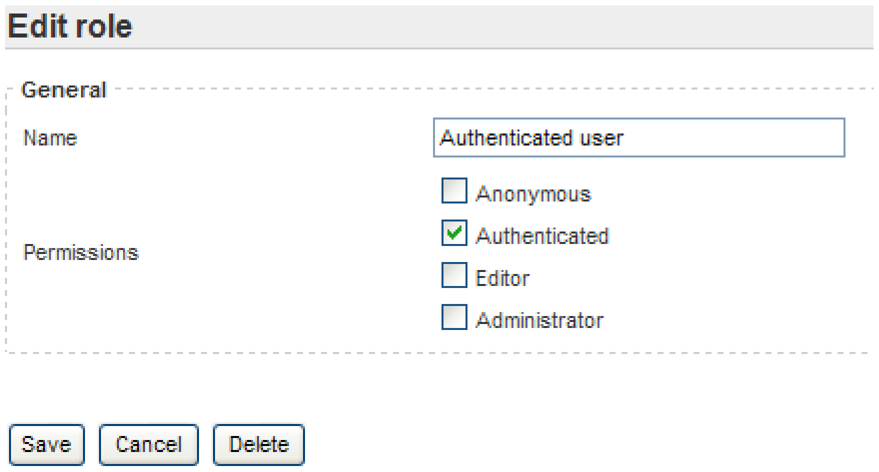
The roles have great responsibility in the security of the content management system. In the figure 38 one can see the webpage for roles administration. Here one can observe the current roles inside the system. Next to each role there is button named edit to change the properties of the current role. There is also another button named add new role, this button helps to define the new roles in the current system.


Roles

	Rolename	Permissionlevel(s)	Last update	
🔒	Anonymous user	Anonymous	7/16/2004 9:18:09 PM	
	Authenticated user	Authenticated	6/25/2004 12:59:02 AM	Edit
	Editor	Authenticated, Editor	6/25/2004 12:59:08 AM	Edit
🔒	Administrator	Authenticated, Editor, Administrator	9/19/2004 5:08:47 PM	

Figure 38: Roles Administration

The figure 39 shows the webpage, which appears after pressing the button to add new role in the system. Here one can assign the name of the role and its corresponding permissions. These are checked by using the check box against permission name. Upon successful assignment of corresponding values the operation needs to be saved using the save button. The deletion of the role can also be achieved by using the delete button. Similarly one can cancel the operation using the cancel button.


Edit role

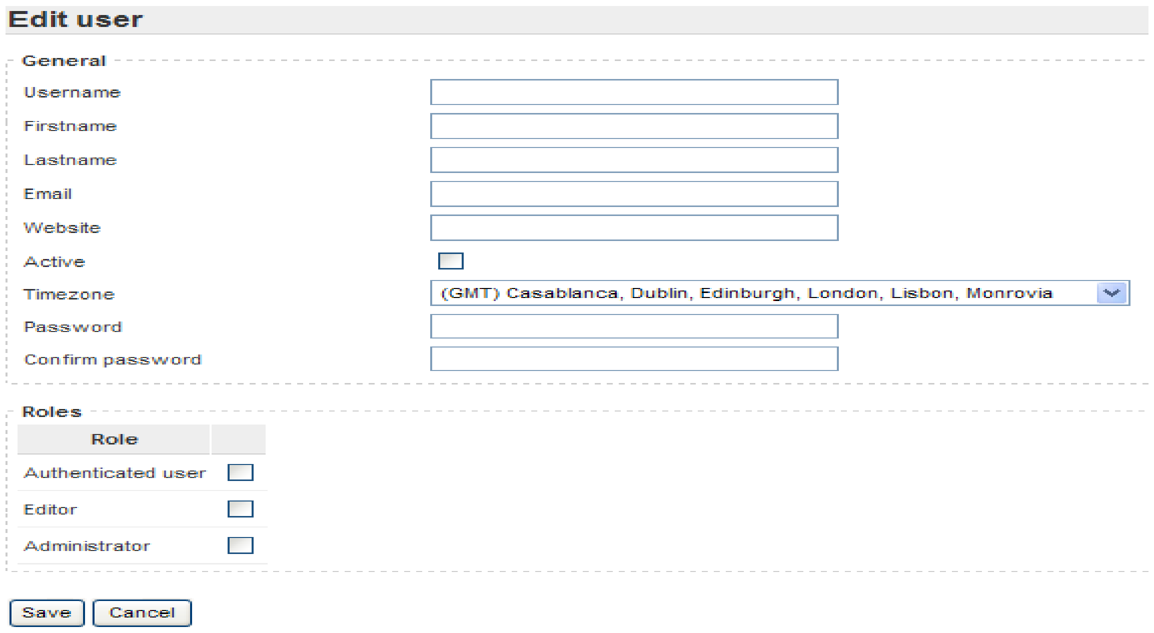
General

Name	Authenticated user
Permissions	☐ Anonymous ✓ Authenticated ☐ Editor ☐ Administrator

Figure 39: Editing the role

6.12 User Management

The figure 40 shows the webpage to manage the users. The addition of new users in the system is carried out by using the user management screen. Here one needs to add the name of the user, email, time zone and password. The corresponding roles for the user can also be given by using check boxes against each role. Upon successful completion of editing the values one needs to be save the values in order makes the action effective.


Edit user

General

Username	
Firstname	
Lastname	
Email	
Website	
Active	☐
Timezone	(GMT) Casablanca, Dublin, Edinburgh, London, Lisbon, Monrovia
Password	
Confirm password	

Roles

Role	
Authenticated user	☐
Editor	☐
Administrator	☐

Figure 40: Managing the Users

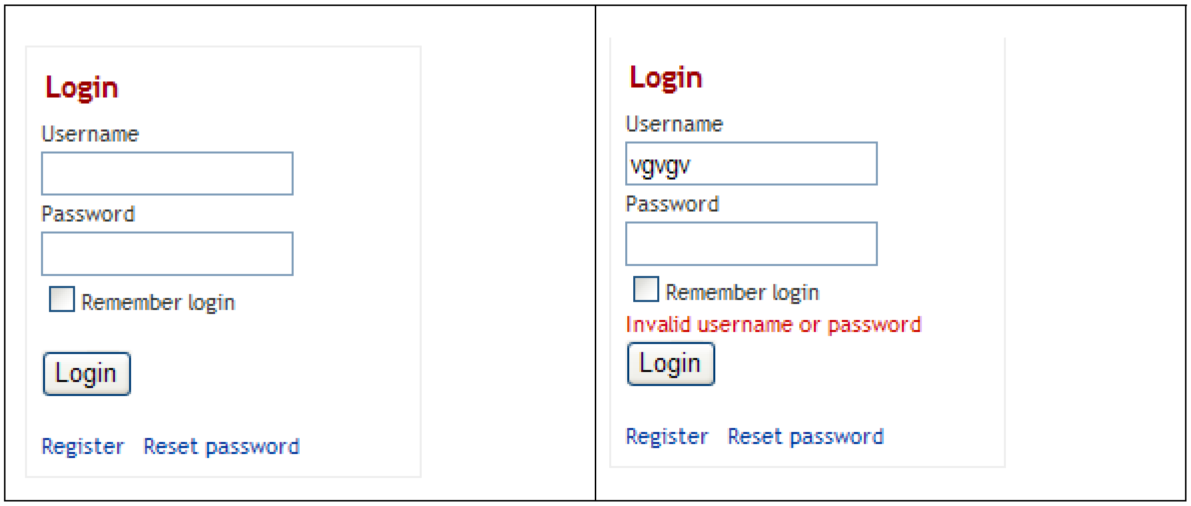
CHAPTER 7: TESTING

7.1 Functionality Testing

In this section, the functionality testing of the software application is shown. The principle components of the total application are cross checked against the expected functionality. The main part of the document management system is classified into three principle components. First one is folder panel, which shows the hierarchical list of the folders in the system. The second component is document panel, which shows the list of documents in the system along with the buttons to download and to access more information about the document. The third component is detail view panel, where one can see the more detailed information about the documents. All these three are very major components of the system. Functionality test is thoroughly conducted on these components. Apart from them there are also other components like login page, document upload page and roles editing page, which are also tested functionally.

Login page

Upon starting the application login screen appears where one needs to provide the username, password details. Upon checking the authentication the system permits the user to access the document management system. The figure 41 shows the login page of the document management system



Login
Username
Password
Remember login
Login
Register Reset password

Login
Username
vgvgv
Password
Remember login
Invalid username or password
Login
Register Reset password

Figure 41: Login Screen

The functionality of the login page is checked by providing the wrong username and password. The system clearly identifies the issue and shows the message as “Invalid username or password” as shown in the right side of figure 41.

Folder panel

The folder panel must clearly shows the parent and child relationship between the folders. This functionality is checked across the sample data present in the database. The system is clearly reflecting the actual view of the folders and their relationships as shown in the figure 42.

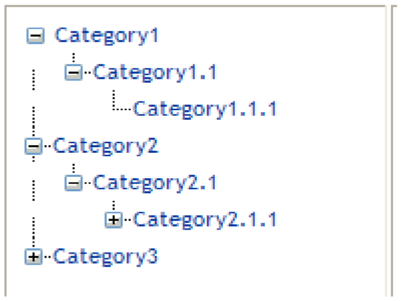
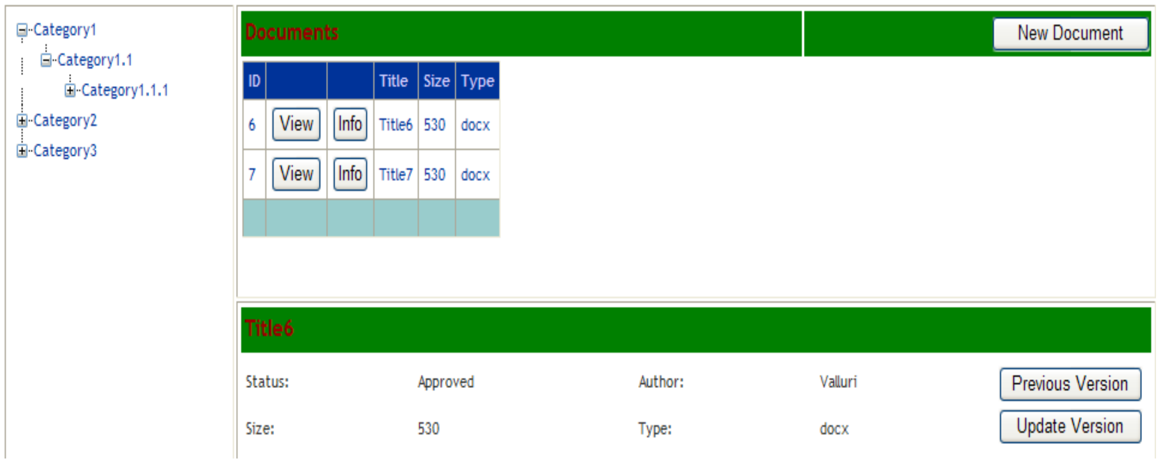


Figure 42: Checking folder panel

Document Panel

When the user selects the respective folder then its contents must be loaded on the document panel. This functionality is cross checked and it behaved well as expected. The list of documents is also shown in the document panel along with the buttons to download the document or to see more information about the documents. All these functionalities are working as expected which can be shown in the figure 43.



- Category1
  - Category1.1
    - Category1.1.1
- Category2
- Category3

Documents | New Document

ID			Title	Size	Type
6	View	Info	Title6	530	docx
7	View	Info	Title7	530	docx

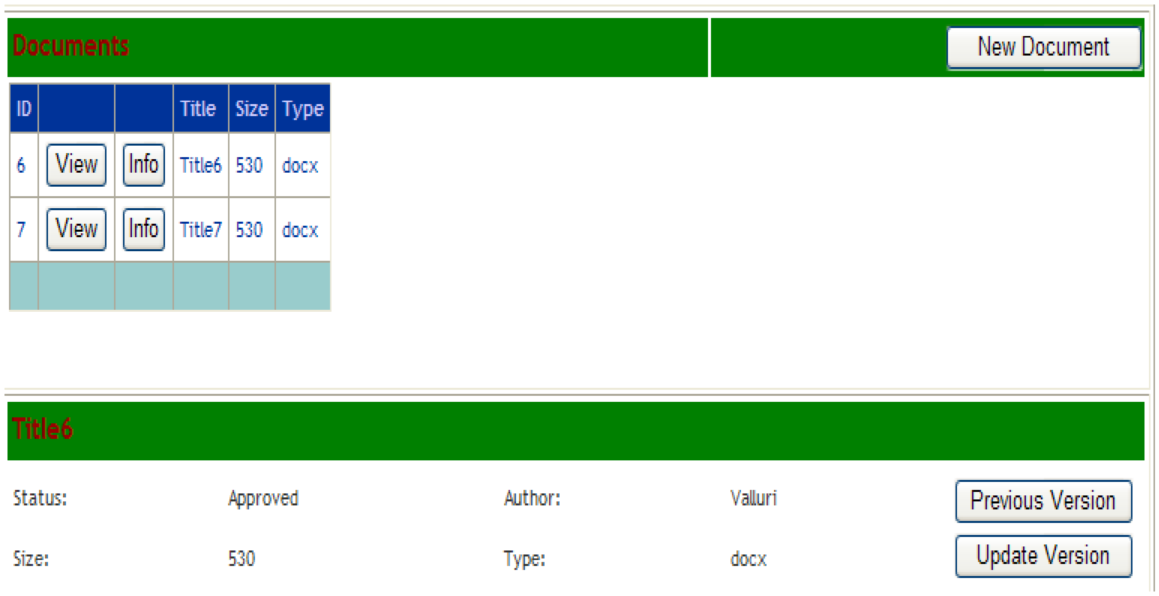
Title6

Status:	Approved	Author:	Valluri	Previous Version
Size:	530	Type:	docx	Update Version

Figure 43: Checking document panel

Details-View panel

Upon selecting the *"Info"* button from the document panel the system must load the Details view panel with the data corresponding to the respective document. The system behaves well in this case and the result is shown in the figure 44.



Documents | New Document

ID			Title	Size	Type
6	View	Info	Title6	530	docx
7	View	Info	Title7	530	docx

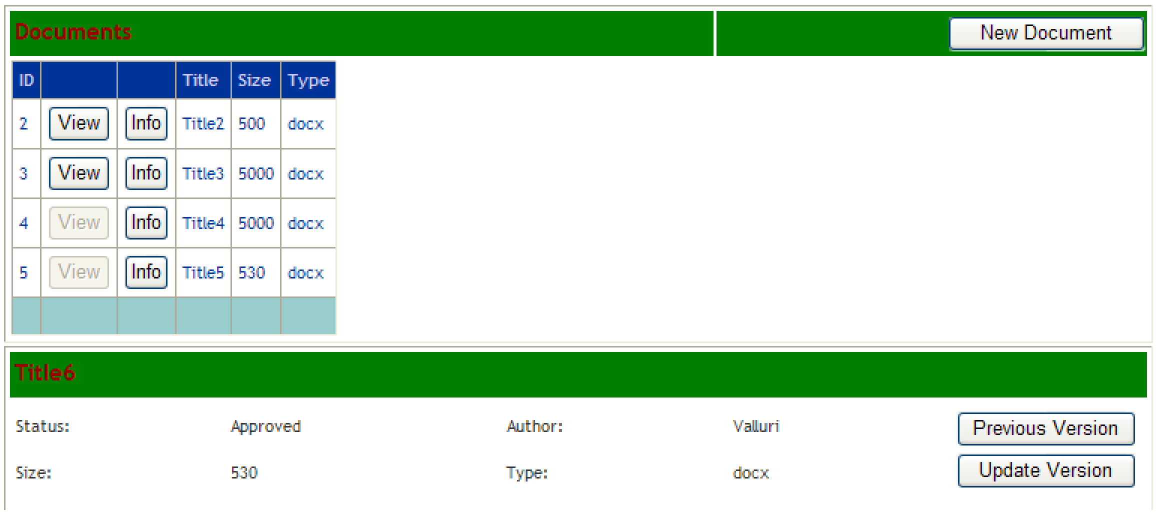
Title6

Status:	Approved	Author:	Valluri	Previous Version
Size:	530	Type:	docx	Update Version

Figure 44: Checking details-view panel

Checking rights

The major concern is to check the access control of the system. System implemented Role Based Access Control (RBAC) mechanism to control the access to the system. The system checks the role of the respective users and changes the functionality according to his credentials. The major important aspect realized in the application is to have control the access in document wise rather than just folder wise. In many applications the access is controlled via just folders, which means the rights are fixed in a way that the user can have access to the contents of the entire folder or not at all. But in this system rights are controlled both from the view of folders and also documents this is achieved by having a better realization of the folders virtually in the database rather than actual creating them in the file system. The functionality is checked and it is behaved correctly in the application as shown in the figure 45.


Documents

New Document

ID			Title	Size	Type
2	View	Info	Title2	500	docx
3	View	Info	Title3	5000	docx
4	View	Info	Title4	5000	docx
5	View	Info	Title5	530	docx

Title6

Status:	Approved	Author:	Valluri	Previous Version
Size:	530	Type:	docx	Update Version

Figure 45: Checking Rights

Updating the version and downloading the document:

The functionality of updating the version and download of the document is checked and it clearly worked as required. The update page is shown in figure 46 and one needs to press the *"Browse"* button to locate the files from the local system and then press the

"Upload" button to send the file to the server. This functionality is clearly behaved as expected.

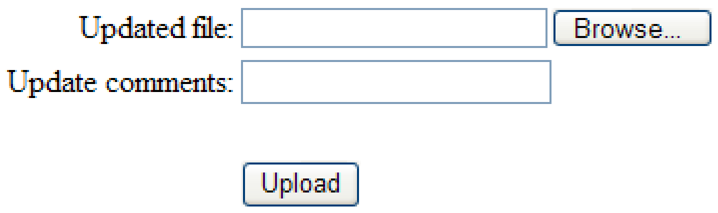


Figure 46: Document Update page

7.2 Unit Testing

A unit test can be described as something that tests a specific unit of the program. It thoroughly tests all aspects of every entity in the application. An example of this would be test for providing that a class works as expected. It is preferable to run the unit tests in an isolated environment. This means when testing one object, one must reduce the dependency on other objects as much as possible, or remove all dependencies preferably.

The total application logic is tested from the beginning of the development and refactored the code several times until it achieves the desired functionality. The different aspects that are concerned in the application as part of unit testing are as follows (33):

- *Boundaries*: Boundary checking is a way of testing the application can handle itself. Suppose the object handles particular range of values as an input, if it receives beyond those range of values then it should know itself handling the situation. This test is to make sure that the code handles itself with regards to the error.
- *Success and failure*: The test is a way to prove the code works as it should. In the first step the test is written to make it pass. Then the second step is to write a test that should fail. This has set of bounds for the application proving what works and what doesn't.
- *Functionality*: The tests are written in order to test the details of the specific functionality that the widget should perform. For instance checking that data layer connect to the database or whether the uploading of the file done correctly.

The application is clearly developed by using unit test in order to have consistency and better productivity of the code. Unit testing avoided the occurrence of bugs at every stage of development.

7.3 Evaluation of the Results

The following evaluation is done on the results obtained:

- The login page is working as it is expected and clearly reacting in the error situations
- The folder panel is clearly reflecting the parent and child relationships of the folders along with its contents
- The documents panel is loading with the list of documents correctly upon selecting the desired folder
- The details-view panel is clearly reflecting the further information about the document selected on the documents panel.
- The uploading screen is clearly functioning and the desired document is sent to the server.
- The downloading of the document is functioning well as expected.
- The document rights are clearly reflected in the user interface by deactivating or activating the desired options.
- The loading of the document in the Subversion version control system is carried out as expected.

CHAPTER 8: FURTHER WORK

8.1 Implementing Rule support for Role based access control

Role based access control plays key role in today's security mechanism. In general roles consists explicit authorizations. Big organizations may need quite large number of roles. The role based access control reduces lot of administrative work, but still it needs some considerable work to assign the roles to users. Rules are well-known technique for automation. The access rights can be grouped using the rules instead of roles. Normally rules consists a condition on the left hand side and one or more actions on the right hand side, when the expression on the left hand side is true then the actions on the right hand side will occurs.

Although rules have greater advantage of automation, there are some disadvantages due to its highly dynamic nature. It is difficult to obtain an overview about who is allowed to do what. It is also difficult to maintain rules in longer run as it is difficult to foresee the impact of the given rule. The approach of combining rules with roles brings the finer mechanism by reducing the disadvantages of the both schemes. In this Rule-based provisioning of Role Based Access Control (RBAC), one doesn't need to explicitly assign the roles. Instead rules using user attributes compute the role assignments of users dynamically. This can be achieved automatically by taking the information from the human resource databases, corporate directories or other information bases in the enterprise (17).

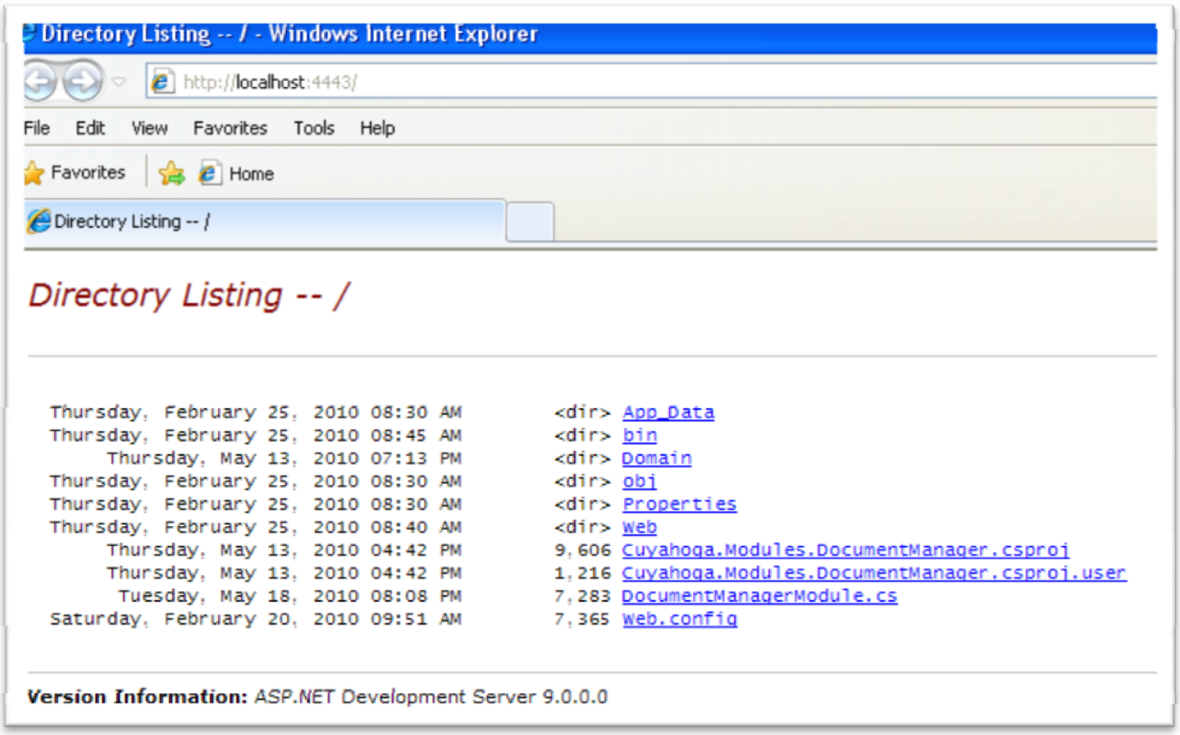
The key advantages of the Rule based provisioning of Role Based Access Control can be described as follows (17).

- This has the major advantages of the role based access control like role hierarchies.
- Rules can be used for dynamic role assignments based on the user's information. This greatly reduces the administrative work associated with assigning roles statically

8.2 Removing reported bugs

Directory Listing: Upon running the project, sometimes login page is not visible instead the lists of web files are shown in the internet browser. One needs to click the starting page explicitly in order to view the required webpage. This must be fixed in order to have redundancy on production server. This bug can be clearly shown on the figure 47.

Upload page link: The link between button named update version in the document management system main page and the upload document page is broken. There is redirection problem existed, which needs to be rectified.



Directory Listing -- /

```
Thursday, February 25, 2010 08:30 AM        <dir> App_Data
Thursday, February 25, 2010 08:45 AM        <dir> bin
     Thursday, May 13, 2010 07:13 PM        <dir> Domain
Thursday, February 25, 2010 08:30 AM        <dir> obj
Thursday, February 25, 2010 08:30 AM        <dir> Properties
Thursday, February 25, 2010 08:40 AM        <dir> Web
     Thursday, May 13, 2010 04:42 PM        9,606 Cuyahoga.Modules.DocumentManager.csproj
     Thursday, May 13, 2010 04:42 PM        1,216 Cuyahoga.Modules.DocumentManager.csproj.user
      Tuesday, May 18, 2010 08:08 PM        7,283 DocumentManagerModule.cs
Saturday, February 20, 2010 09:51 AM        7,365 Web.config
```

Version Information: ASP.NET Development Server 9.0.0.0

Figure 47: Start page Bug

8.3 Remaining Features Implementation

List of Revisions: It is nicer to have a list of all the available versions of a particular document apart from the latest document. This makes easier to select the required version of the document by the user.

Base lining: It is highly advised to have a base lining feature of the documents, where all the documents up to the particular quality gate can be automatically downloaded and bundled into the zip folder upon request. There can be also feature to send this documents folder through the email of the desired client.

Viewing the Document in the Web browser: Currently documents are downloaded to the user's computer upon request. Sometimes user may not have the software application to view the downloaded document. It is nicer to have a feature to see the document directly in the web browser.

Search feature: Currently the search feature is not attached to the document management module. This feature is actually resided as a service in the framework. One can easily attach the document search feature to the application. There is no need to develop this feature.

8.4 Browsers Compatibility

The web application is checked on internet explorer and Mozilla Firefox. The users may use many different browsers to have an access over the system. It is advisable to check the functionality of the web application in many other browsers available on the market. This compatibility tests will help to avoid any possibility of malfunction in any browser before deploying to the production server.

8.5 Measuring Stress of the Web application

One must employ good optimization methods to handle the stress on the web application due the high number of concurrent users. There are many tools available on the market to measure the stress level test on the web server. Without having proper planning about the usability of website it may leads to the slower performance in the access of web pages. At the end of the road performance plays key role in the success of any web application.

CHAPTER 9: CONCLUSION

In this project a web application is designed and implemented for storing and retrieving the documents along with their metadata by having security mechanism as role based access control. The system stores the documents of the users along with the document and user's information. This is created as a web application, where users can access to their documents from their local system irrespective of their location with the help of wide varieties of web browsers. The total system is carefully designed and implemented using sophisticated security architecture with the help of role based access control mechanism. Every component of the total system is carefully selected depending on many criteria's like cost reduction, extendibility and maintainability. During the whole process of development the following tasks are accomplished.

The total database is divided in to two parts having subversion for the actual documents and the relational database for the metadata corresponding to the documents and the user's information. This division clearly helps for having better search over the system. The storage space complexity involved in storing the documents are clearly solved using the versioning control system named subversion to store the raw documents.

The database persistence layer is realized using data persistence engine named as NHibernate, which brings the capability to use most of the relational databases upon selection. Layering concept helps for the cleaner separation of the code. The total system is designed on enterprise scale according to the domain driven design principles in order to attain better future extendibility and maintainability.

The system's security mechanism is implemented using widely accepted Role Based Access Control mechanism. This mechanism is well suited with the requirements of the system. The total access control mechanism is realized practically according to the application needs.

The tree view of the folders is virtually created from the values in the database rather than really creating the folder in the file system of the server. This technique clearly helps for better maintainability of the parent and the child folders.

The total list of documents along with their metadata is clearly visible in a grid view to the user via the friendly to use user interface. The available functionality or options of the user interface is dynamically changed according to the calculated roles from the user’s credentials.

At each and every stage of development unit-testing is employed in order to employ test-driven development techniques.

Further work involves solving the reported errors, implementing remaining features and browser’s compatibility is clearly specified in the chapter named further work.

REFERENCES

1. *Role Based Access Control for the World Wide Web.* **John F. Barkley, Anthony V. Cincotta,David F. Ferraiolo, Serban Gavrilla,D. Richard Kuhn.** 1997. p. 2.
2. **MATHEU, NURIA FORCADA.** *Life Cycle Document Management System For Construction.* 2005.
3. Document Management System. *wikipedia.* [Online] [Cited: November 10, 2009.] http://en.wikipedia.org/wiki/Document_management_system.
4. *AIIM.* [Online] [Cited: Febraury 5, 2010.] http://www.aiim.org/.
5. *CONTROL MECHANISM FOR INFORMATION SHARING IN AN INTEGRATED CONSTRUCTION ENVIRONMENT.* **M. Sun, G. Aouad.** 1999. p. 4.
6. **Control, Role-Based Access.** *David F. Ferraiolo, D. Richard Kuhn, Ramaswamy Chandramouli.* s.l. : ARTECH HOUSE, 2007.
7. ALPINE mobile media solutions. *ALPINE GERMANY.* [Online] http://www.alpine.de/.
8. *INotion.* [Online] https://inotion.alpine.de/.
9. I-logix. *Wikipedia.* [Online] [Cited: December 10, 2009.] http://en.wikipedia.org/wiki/I-Logix.
10. Alpine Selects I-Logix iNotion Portal for Process Improvement Project. *Embedded star.* [Online] [Cited: January 15, 2010.] http://www.embeddedstar.com/press/content/2004/9/embedded16295.html.
11. *An Adaptive Document Version Management Scheme.* **Boualem Benatallah, Mehregan Mahdavi, Phuong Nguyen, Quan Z. Sheng, Lionel Port, Bill McIver.**
12. Subversion. *Subversion.* [Online] [Cited: November 29, 2009.] http://subversion.tigris.org/.
13. Comparision of revision control software. *wikipedia.* [Online] [Cited: December 20, 2009.] http://en.wikipedia.org/wiki/Comparison_of_revision_control_software.
14. Intro to Distributed Version Control (Illustrated). *Better Explained.* [Online] [Cited: January 5, 2010.] http://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/.
15. **Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato.** *Version Control with Subversion.*
16. *Role-Based Access Control.* **Ferraiolo, D., and D. R. Kuhn.** 1992.
17. *Rule Support for RoleBased Access Control.* **Axel Kern, Claudia Walhorn.** 2005.
18. iBATIS, Hibernate, and JPA: Which is right for you. *Javaworld.* [Online] http://www.javaworld.com/javaworld/jw-07-2008/jw-07-orm-comparison.html?page=6.
19. *Microsoft .NET.* [Online] http://www.microsoft.com/net/.
20. *dotnetnuke.* [Online] [Cited: December 20, 2009.] http://www.dotnetnuke.com/.
21. *CUYAHOGA FRAMEWORK.* [Online] http://wiki.cuyahoga-project.org/BaseArchitecture.ashx.
22. *Apache HTTP Server Project.* [Online] [Cited: December 3, 2009.] http://httpd.apache.org/.
23. *NHibernate Forge.* [Online] [Cited: November 15, 2010.] http://nhforge.org/Default.aspx.
24. *PostgreSQL.* [Online] [Cited: November 2, 2009.] http://www.postgresql.org/.
25. *castle project.* [Online] [Cited: October 20, 2009.] http://www.castleproject.org/.

26. *ASP.NET.* [Online] http://www.asp.net/.
27. **Seddighi, Ahmad Reza.** *Spring Persistence with Hibernate.* s.l. : PACKT Publishing, 2009.
28. **PIERRE HENRI KUATE, TOBIN HARRIS, CHRISTIAN BAUER, GAVIN KING.** *NHibernate in Action.* s.l. : Manning, 2009.
29. *The C# Language.* [Online] http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx.
30. SharpSvn Namespace . *sharpsvn.* [Online] [Cited: November 8, 2009.] http://docs.sharpsvn.net/current/.
31. **Nilsson, Jimmy.** *Applying Domain-Driven Design and Patterns: With Examples in C# and .NET.* s.l. : Addison Wesley Professional, 2006.
32. Model-view-controller. *wikipedia.* [Online] [Cited: December 10, 2009.] http://en.wikipedia.org/wiki/Comparison_of_revision_control_software.
33. Appendix B Test-Driven Development and Continuous Integration. *Packt Publishing.* [Online] [Cited: Febraury 4, 2010.] https://www.packtpub.com/sites/default/files/4787-Appendix-B-TDD-and-Continuous-Integration.pdf.

APPENDIX

This Master report contains an appendix of Master thesis in PDF-format, Source Code, Sample Database and Installation help file on a CD. This Appendix is deposited with Prof. Dr.rer.nat Hans-Jürgen Hotop

Declaration

I/we declare within the meaning of section 25(4) of the Examination and Study Regulations of the International Degree Course Information Engineering that: this Master report has been completed by myself/ourselves independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, June 7th 2010 ---
Subhakara Valluri