School of informatics
University of Manchester

Web based document processing and management system

Lin Sun
BSc (Honors) in Computing Science

Project Supervisor: Dr. XiaoJun Zeng

Final Year Project
Final Report
19 April 2006

Abstract

The aim of the project is to develop a document management system that is able to deliver access to anyone authorized anytime, anyplace on any device. The system developed makes use of rich internet technology to replace desktop application with rich internet application. The system shares the advantage of both web application and desktop application, and removes the most disadvantages of both. The problems solved in this system includes

Absence of support
No support for a document type simply means no access to the content of the document.
Complexity of various software packages
Different document type use different software package which work in the way differ dramatically, which significantly reduce the usability.
Security
Virus and Spy ware hidden in the document
Maintenance
The longer the lifetime of software, more bugs will be exposed. Update and configuration on each client can be a headache.

At the end of the project, a web based document management system is developed. The built-in document type supports are Microsoft Word, PDF, and Postscript. This system enable user to manage document within the web browser and without any support installed. In other word, the only requirements are an internet connection and a web browser.

Table of Content

LIST OF FIGURES

1. Introduction

1.1. Mission Statement

The project aims to develop a system is for any type of document but nothing particular. This system is revolutionary to document processing. The uniqueness is being web based and for general-purpose. Being web based makes this system available everywhere through internet. Being general-purpose means system can be used for all type of documents and for most organizations.

1.2. Project Overview

The project causes a revolution of the way people work with document. Before this revolution, people think working with document means working with word processor. However, new web technologies make web powerful enough to simulate an operating system and even a word processor. People no longer need to work with a specific word processor. Using the proposed system, user can work with any document everywhere.

Why this project?

- To break document type barrier, all document type edited within browser in same way
- Web based, no software installation on client
- Centralised data and processing program, easy to maintain
- No high profile similar product on the market

What technologies are used?

JavaScript, HTML, AJAX(Asynchronous JavaScript language and XML), C#

The principle of the system is:

- Keep it Simple
  - Extensibility
  - Use-at-will architecture
- Showcase of current trend in web development
- Portability (develop once, run on all platform)
- Open Source (to suit the nature of this project)

1.3. Benefits

The proposed system is expected to revolute the way of document processing. It will bring benefits to the final user.

From the user perspective:

Edit document independently, more flexible

“Independently” here means being free from the influence, guidance, or control of the working environment. The only requirements are a connection to the server (internet), and a web browser. No download, no installation and no manual configuration.

All document type in one editor, easier to use

The proposed system will aim to provide platform to support all document type. User can edit all document type in one system in the same way. In contrast, OpenOffice and Microsoft Word work

in different way, and user need to be trained to use both systems.

No software installation, more efficient, more secure

From the user's perspective, this means the user can get their hands on the document much quicker. User can start working with the document when sitting in front of any computer, and do not have to worry about the working environment. In addition, this system consumes less resource of the client computer. Microsoft Word typically uses memory over 30MB. (Gsurface, 2004) A web browser typically uses memory below 20 MB or even less. Besides, almost all security issues for client side are also solved by this approach. Because all operations are executed with browser with nothing installed, no information except cookie and cache can be written to the client hard disk by JavaScript in principle. Most of the security concerns like virus or spyware can not work even it are unknown. For example, a document contains a virus, which will function when user open the document. By using the system, firstly the virus can be possibly killed by the antivirus firewall on the server; even if the virus is unknown and it is not killed by the server, it still cannot function because every document has to be rendered to HTML in this system. User actually open the HTML file, not the original file.

From the administrator's perspective

Portability, suit the current infrastructure

To install the proposed system, no change need to be made on the current infrastructure, the system will suit most infrastructures automatically. The system will run on a number of major operating system and architectures, including Windows and Linux.

Maintenance cost reduced

In the proposed system, software will only be installed on the central server, and nothing needs to be installed on the client side. In contrast, traditional word processor will require user to install a single copy on every client. To maintain these single copies can be a complex task. In the proposed system, most maintenance work can be done one time on server side. A considerable amount of cost can be saved by working with server side only.

Customizable

With a user system and a file system that are both a simulation of traditional operational system, and a plug-in based document processing system, administrators can build a specific system for their own requirements.

1.4. Challenges

The proposed system is challenging and there are few similar systems on the market. In this section, the challenging aspect of the project will be stated.

Most difficulties of this project are from the client side, namely the HTML editor. The HTML editor is completely implemented with JavaScript. Javascript as known, it is a scripting language. Its function is very limited. JavaScript is not designed for the large scale application like this project. Additionally, different browsers will render the same JavaScript code differently. This makes writing a cross browser Javascript program a very difficult task. As a proof, a similar project "Writely" is said to be at the front end of Javascript, and a Html Editor wiki is also said to be at the front end. (Michael,2005) The proposed system brings these two front ends together, which will stand on the frond edge of JavaScript programming.

Some technologies used in the proposed system are quite new. Like "AJAX", which just started to

be used by Google at the time of writing, is a revolutionary technology for client and server communication in web programming. It is explained in detail in “technology specification” section. Technology like “AJAX” does not have many high profile implementations at the time of writing. The project itself will dedicate to be a high-profile product with usage of new technologies. Another challenge is the web operating system. Despite the term WebOS is out for more than one year, there is still not even a basic “Window” control. In other word, this system needs to be developed from its Framework. (Imagine developing a window application starts from developing a window). The difficulties in implementation are also enormous. It is stated in detail in the implementation section.

To sum up, the project is a challenging project because of the limitation of Javascript, the new technologies and the shortage of Web operating system framework. These challenges even result in open issues that cannot be solved in this project which are discussed in the open issue section.

1.5. Development Methodology

A phased development methodology will be used in this project. In the process, various versions of the system will be produced. This methodology can give user a chance to work with the system sooner, and give chance to identify additional requirements. (Alan Dennis, 2002)

Firstly, the background of the system will be studied. The system with similarity will be examined, and their advantages and disadvantages will be listed. Secondly, the analysis and design will be carried out. Before implementation, only the analysis and design of the core function will be carried out. As the nature of phased development, it will bring function to final users more quickly to identify additional requirement. Therefore, before implementation of each part of the system, the analysis and design will be reviewed. Implementation phase will be divided into three parts according to the structure of the system: Client Side development, Server side development and Database development. A prototype system will be developed. The final stage is testing and documentation writing. A test report will be produced. The test is composed of Unit testing and Integration testing. The unit test contains black box testing and white box testing. The integration testing contains the use case testing, use interface testing and interaction testing. Furthermore, program documentation will be produced. Documentation includes developer manual which guides any further development, user manual which gives explanation on every function, and tutorials which will guide the usage of the system.

1.6. Success Criteria

How this system can be judged as a success? These are the general criteria. For the detail requirement, please refer to requirement analysis section. Solely with this web based system:

- User can read PDF, PostScript, and Microsoft Word Files.
- User can edit Microsoft Word documents.
- User can perform basic file/folder manipulation.
- The edited word document is compatible with the original Microsoft Word processor..
- User can share document to other people. User can browse all documents that available to him and edit the shared document if permitted.
- A familiar interface for the user with no previous experience.

2. Background

In the section, firstly the driving trends in software development is briefly discussed, then the nature of how human work with an electronic document will be investigated, then five types pre-existing system will be reviewed. Their strengths and weaknesses will be listed at the end of the section. Finally, how these systems help to build the proposed system will be examined. As a conclusion, a blueprint of the proposed system will be stated, and a diagram of the proposed system will be drawn.

The Driving Trends in Software Development

The 11 years old “information at your fingertips” speech from Bill Gates proposed that

“Today's CD-ROM and online services are wonderful examples of software that prepares us for the possibilities of the future. Imagine the best of both mediums combined and running on a high-bandwidth, high-speed network: high-capacity, shared storage that enables up-to-date, rich, multimedia content to be easily accessed by many people. And ***by 2005****, there will be applications that relate to all aspects of our lives.”(Bill Gates,1994)*

At the time of this report, it is 2006, and Bill Gate’s prediction has recently been now realized as *Applications Able to deliver Access to Anyone Authorized Anytime, Anyplace on Any Device.* This is called the eight A’s principle, and these words are almost the most keyword advertised in the software industry nowadays.

SAA (System Application Architecture) from IBM is arguably the first attempt to comply these rules. SAA was IBM's strategy for enterprise computing in the late 1980s and early 1990s. (now expanded by Open BluePrint) By using this standard, an enterprise could be assured of consistency and some amount of program portability among IBM's range of computer systems.

It is still a challenge now in the software development to develop the 8A’s software distributed over n-tiers as simply as possible. The requirement from the 8A makes the software development to become increasingly complex, because 8A’s rule imply that the application is not confined in one machine but distributed into many tiers. It is required that application is available not in one system but to most systems with similar interface. With regard to the “any device” principle, the effort made by the GUI designer is oppugned because the interface is required to display gracefully and usably on many different display modes. Secondly, the application’s transaction (life) extends overtime, the longer it exists, the more risk of exceptions, conflict and required disengagement (update, patch). Furthermore, these requirements are further augmented by the security concerns which add even more complexity.

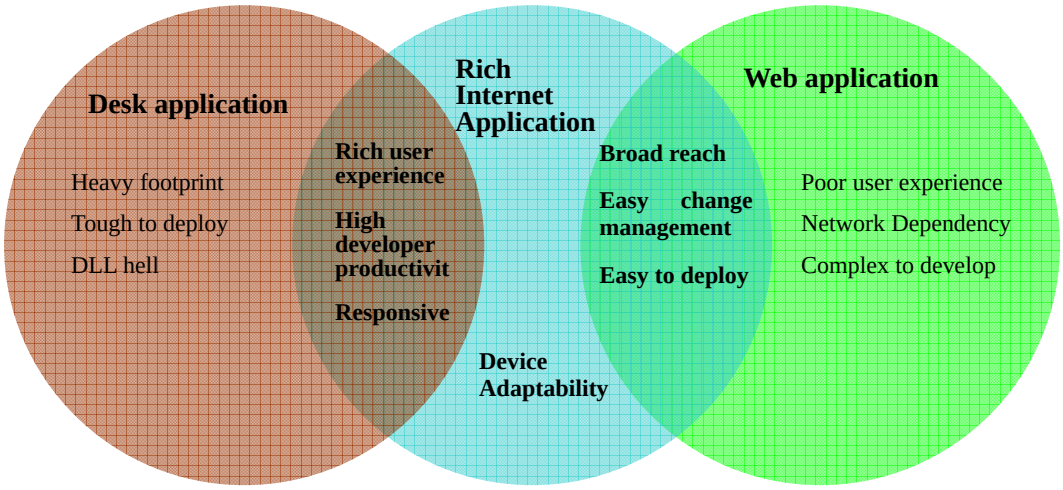
Rich internet application

The rich internet application is a simple way to achieve 8A. The rich internet application describes an online application or utility that includes a level of functionality and interface complexity formerly ascribed only to desktop applications. Ajax as one of the most recent web 2.0 rich internet technology is used through the project. The applications use well-documented features present in all major browsers on most existing platforms. At the time of writing, Ajax applications are effectively cross-platform solution.

Ajax deliver content in the form most appropriate to the message, as it use DHTML/XML as the primary message carrier. It makes development of the interface an easier task because HTML is friendlier to information than the typical programming language. For example: How much effort do you need to represent the following HTML in JAVA GUI?

Bold Font

Ajax application as one type of rich internet application shares the advantages of both typical desktop client (heavy client) and web application client(thin client).



Comparison among desktop application, rich internet application and web application

Particularly, compare to the desktop application, Rich internet application including AJAX shares a number of advantages:

- No client installation required, use can start use a program immediately.
- Users can use the application from any computer with an internet connection, and mostly platform independent.
- Web-based applications are generally less prone to viral infection than running an actual executable
- If there is a web alternative version available, users are unlikely to install new software.

Compare to the traditional web application, the rich internet technology eliminates the start-stop-start-stop nature of interaction on the Web by introducing an intermediary layer between the user and the server. Use Ajax as example, instead of loading a webpage, at the start of the session, the browser loads an Ajax engine which is written in JavaScript. This engine is responsible for both rendering the interface the user sees and communicating with the server on the user's behalf. The Ajax engine allows the user's asynchronous interaction with the application to happen independent of communication with the server.

2.1. Process of working with a document

A document here is considered as a computer electronic document, not the paper document. The process of manipulating a document normally involves 4 unique stages: Create, View, Edit and Save. The last three stages usually tend to be repeated. Therefore, the process can be generalized to a process that firstly a document is created and then it might be edited, read or saved for numerous times. If a document is published, View would be the most frequent action performed

on a document.

2.2. Word Processor

To perform these stages, a working environment called Word Processor is normally required. The most frequently used word processors on the market are Microsoft Word, OpenOffice, AppleWorks. Many concepts and ideas are originated from ‘Bravo’ which is the world first ‘*WYSIWYG document preparation program*’ in 1974. It provided multi-font capability using the bitmap displays on the Xerox Alto personal computer (Bulter, 1979). These features served as the basic idea of WYSIWYG. WYSIWYG is to make the user to be able to visualize what they are doing. In other word, an instant change of display will be given in response to a user action. The WYSIWYG is the most important feature of contemporary word processor. Such word-processor is composed of document editor and document management system. View and Edit can be done using the document editor; Create and Save can be done with the document management system which cooperates with the file system.

In a recent poll done by Desktop Pipeline, asked participants to vote and comment on their favorite word processor, the results favored Microsoft Word by quite a large margin. Sixty-one percent of respondents use Microsoft Word, while thirteen percent use WordPerfect. Four percent of respondents use a text editor as their primary word processor and an additional seven percent use “other” programs. Sixteen percent who use open source word processors such as OpenOffice.org Writer or StarOffice. (Desktop Pipeline, 2005)

I'm a traditionalist: I use Microsoft Word.	
	**58%**
I'm a real traditionalist: I use WordPerfect.	
	**12%**
I'm a minimalist: I use a text editor.	
	4%
I like open source: I use OpenOffice.org Writer or StarOffice Writer.	
	**18%**
I use something else:	
	**7%**

Table 1 “Which word processor do you use?” (Desktop Pipeline, 2005)

A problem emerged with the diversity of different word-processors: Cross-Platform. The document type of Microsoft Word does not work cross platform. However, Microsoft promises to bring “save as pdf” feature in Office 2005, which allows user to read document on the other operating system. At time of writing, its competitor OpenOffice can run on all major platforms, including Microsoft Windows, GNU/Linux ("Linux"), Sun Solaris, Mac OS X (under X11), and FreeBSD.(OpenOffice,2005) Additionally, there are many tools to help cross-platform view/editing. In general, the cross-platform problem is partly solved or compromised on the premise of having a kind of working environment installed.

A broader problem is generated with this solution or this problem exists all through: what if a specific word-processor is not installed?

Not having support for a particular document type is frequently encountered. It results in

inaccessible document and no work can be done until word processor software installed, providing it is possible to be installed. The software can not be installed if no software is available or the system environment does not allow the installation. By installing a number of software, users can finally hand on the work, but this is a waste of time and IT resource as stated following.

Waste of IT resource means the cost to manage these programs. Management generally means customization, upgrade, configuration and security. As every computer in an organization has to be installed a copy of the word processor, to maintain these copies are usually a complex task. In order to save the cost and effort, the document processor needs to be centralized. This means no software installed on the client side; all document processing are executed on the server side. How the centralization can be achieved? A client that needs less maintenance is required and it must be widely available. The suitable candidate is an editor in web browser.

2.3. An alternative: WYSIWYG Html Editor

HTML and JavaScript is a built-in feature of every browser. Since Internet Explorer 5, a new command 'object.execCommand' has been introduced. And Firefox adds its support for the command in the release 1.3. "ExecCommand" is used to execute a command on a document that can manipulate the page layout, insert an image, url, list, text box, horizontal rule etc right into a HTML document in the browser. (Paul Abraham, 2003) This makes it possible to simulate a basic WYSWYG document editor. There are more than 80 html editors on the market, some are open source. (Paul Browning, 2005) They are all based on JavaScript and some of them are cross-browser. Cross-browser generally means they can work with different explorers on various platforms. The browsers of the measurement are FireFox, Internet Explorer, Mozilla and Netscape. Cross-browser is always a problem to WYSIWYG HTML Editor. As different browsers will render the same markup differently, which means the same code sometimes can have different effect cross browsers. Moreover, the browser updates quite frequently, and the code might be rendered differently across different version. Finally user setting is also a problem. User setting includes font-size, resolution, contrast etc. All developer can do is to suggest user an appearance.

2.4. Content Management System

Content Management System is for both document management and document editing in contrast to WYSIWYG HTML editor which only has document editing function. Content Management System is mostly used to manage the content of a website. (James Robertson, 2003) It can dramatically reduce the turnaround time of new content for a business. It usually includes the WYSIWYG HTML Editor as part of the system. The shortage of content management is a document processing system. Most content management system can only edit html or plain text, which does not satisfy the requirement of support for advanced document type like Microsoft Word.

2.5. Elicitation from Eclipse

'The Eclipse Platform is an IDE for anything and for nothing in particular.' (Object Technology International, 2003)

Eclipse is an open-source extensible IDE that can be used to support many programming languages. It is considered to be completely platform and language neutral. In addition to the

languages supported by the Eclipse Consortium (Java, C/C++, Cobol), there are also projects underway to add support for languages as diverse as Python, Eiffel, PHP, Ruby, and C# to Eclipse. (Object Technology International, 2003) The Eclipse Platform's principal role is to provide tool providers with mechanisms to use, and rules to follow, that lead to seamlessly-integrated tools. (Object Technology International, 2003)

Theidea is to decompose the role of IDE into a universal platform and different specific language supports. This universal platform adds capability of handling infinite number of languages. And the language support itself can be built by different project done by different people. This feature also combined with the strength of open-source, which means a large number people can join the development of projects and a huge number of plug-ins and language-supports would be developed. According to the definition of open source (Bruce Perens, 1997), the open source software must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

2.6. Writely – A rival

The discussion of this report so far does not describe any “similar system”, but “Writely” is not just similar; it is a “near rival”. According to official website of Writely, “Writely allows you to edit documents online with whomever you choose, and then publish and blog them online.” And it is recently becomes part of Google. The features of the system are listed below:

- Upload document or create new document online.
- Edit document with simple WYSIWYG Html Editor
- Share the document with other people
- Post the document to blog.

From the understanding of the current progress Writely made, most of its features are included in the proposed system. Document history is a unique feature of Writely. The editing history of a document can be viewed and roll back.

Writely mainly focused on the share and publish or even “blog” of the document. From the understanding of the author, the aim of Writely is to produce an online publishing system rather than a document processor. (Writely, 2005) Therefore, it does not make much effort to simulate the traditional working environment, so it does not have a graded user system and a file system.

Moreover, The Writely project is not open source, which means every document processing unit need to be developed by one team. This results in the development of a new document processing unit a slow process. So far, only word document and plain text are supported by the Writely project.

Furthermore, Writely does not have a search facility. User cannot browse all the resources available to him. In other word, only one document at a time can be shared, no multiple file or folder can be shared.

Another very serious technical issue with Writely is that it can destroy the format of the original document. As an experiment, the author uploads, edits and downloads this report using Writely. The original format is totally destroyed by Writely.

2.7. Summary

The word-processor, WYSIWYG Html Editor, Content Management System, Eclipse and Writely have been investigated above. Here are points of the investigation:

- **Word-processor:**
  - Feature: Word-processor is the origin of document processing, and It has most functions required to process a document.
  - Advantage: Support Rich-Text Document, WYSIWYG Interface,
  - Disadvantage: nature of platform dependence, not available all the time, need to be centralized
- **WYSWYG Html Editor**
  - Feature: Basic document editing can be performed within any browser.
  - Advantage: intention and potential of complete client independent
  - Disadvantage: Sometimes not cross-browser, Different User Setting, Bug of browser
- **Content Management System**
  - Feature: It makes use of WYSIWYG Html editor to manage a web system, which has a Client/Server Infrastructure.
  - Advantage: Built-in WYSWYG Html Editor, Client/Server infrastructure, faster turnaround time for new content
  - Disadvantage: Share the weakness of WYSIWYG HTML Editor, Only for HTML file or Plain Text
- **Eclipse:**
  - Advantage: Language neutral, Open Source Project
- **Writely:**
  - Advantage: Document Editing History
  - Disadvantage: Lose Format Information, no file system, no powerful user system, No folder/multiple share

In the proposed project, the advantages of these systems stated above will be unified, and some of the weaknesses will be improved. A word-processor is the one with most document manipulation functions among these systems, so the main idea will be simulating a word-processor function using Client/Server structure. The whole system will be more or less similar to a content management system which has a C/S infrastructure but document processing units are added to support different document type. On the Client Side, a WYSWYG HTML Editor will be used to perform the editing function. On the Server side, a document processing unit is employed to convert the HTML Code to the targeted document encoding or the other way around. The idea and technologies will be discussed in detail next section.

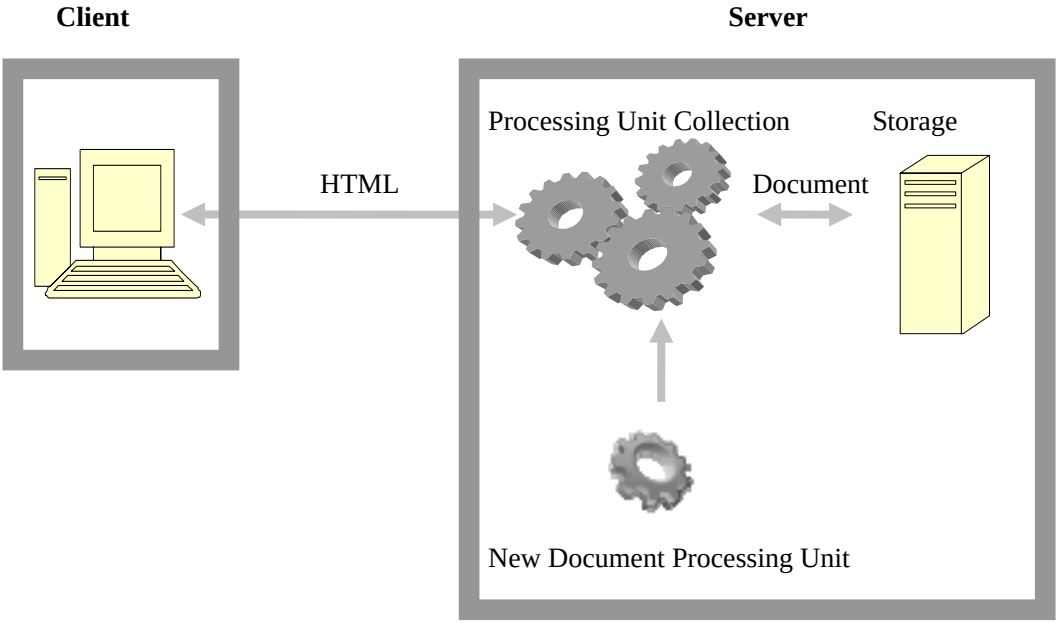


Figure 2.3 Blueprint of proposed system

3. Requirements Analysis

This chapter aim to identify the requirements progressively. Firstly the only stakeholder in the system is identified. Secondly, the requirements from user are gathered. Thirdly, functional and non-functional requirements are identified from the users' requirement. Finally, the technologies used are analyzed in detail.

Please note, all diagrams are generated by IBM rational rose 2003. The internal information is revealed.

3.1. Stakeholders

Some of the principles of this system are simplicity and general-purpose. Thus, generally speaking, only one "type" of user is modeled in this system.

Typically, a web system will divide the user into administration user and general user. However, in this system, administration is multi-leveled. Therefore, every user in the system can have some administration privileges. For example, a department manager can have right to manage the user in the user group of that department; every member in the system can have right to restrict guest to access his document.

The type of user belongs to different user group, and user group can have different permissions. Everyone in this system include guest is in a user group. What user can do totally depends on the permission setting.

3.2. Functional requirement

From the user requirement, the functional requirements are divided into 3 divisions: Document, File and Folder and user system.

Document:

The requirements regards to document are the manipulation and publish of the document.

- The user can open a local document by uploading it to the server. A copy of the local document will be retained on the server with the modified version of the document. [Compulsory]
- The user can create a blank document on the server. [Compulsory]
- The user can view and edit the document and save it or save it as another format.
- The document can be shared to public or a selected group of user or through password protection. [Compulsory]
- Images from internet can be inserted to the document by referencing the web address. [Compulsory]
- Local picture can be inserted to the document by firstly uploading to server. [Compulsory]
- New document processing unit can be easily installed and uninstalled. The system can be easily customized by adding or deleting process units. [Optional]
- User can save the document as an image. [Optional]

User and User group:

Users are graded using user group. Each user is assigned a user space according to policy in user group.

- All users are grouped. Each user group has a level or grade. The user group can manage the user group below its level if permitted. [Compulsory]
- User group can be created, deleted and edited. If a user group is deleted, all user associated with this user group will be deleted. If a user is deleted, the user space associated with the user will be deleted. [Compulsory]
- A user can only have one user group. If a user needs to be in two user group, then a new user group combine the settings of those two groups need to be created first. [Compulsory]
- A user group has a default user space size, and a space exceeding policy. Additionally, user group has a global permission, which is the permission setting used when the permission setting for the user group is not found in the file system. For example, a new user group doesn't have any permission setting in the current file system. A new user group can also be generated by inheriting from another user group. All the settings of the user group will be inherited. But in the file system, the file or folder can have specific setting for the new user group. [Compulsory]
- Every user can have one or more user space to store their resources. The default size is determined by the user group policy. This size can be changed by administrator. [Compulsory]
- If the size of user space is exceeded, the default policy will apply. The policy can warn user only or strictly forbid any exceeding of space size.
- User can search all documents available to him by title, content, date, or owner.

File and Folder:

Files and folder are basic elements of file system. Each file or folder can have a permission setting that associated with user group, and these setting can be inherited.

- Every file must exist in a folder. A folder can have many files.
- Each file or folder has a permission setting for every user group. The setting can be specific or inherited from parent folder.
- User can only perform File or folder manipulation with permission. [Compulsory]
- User can manipulate file/folder include editing, creating, deleting if he has the permission to do so. Every time a manipulation request is made, the permission setting is checked against the user. [Compulsory]
- User can set permissions for a specific file or folder on a user group if permitted. The file or folder can be shared in this way namely setting permission for a user group.[Compulsory]
- File and folder can be shared using a password protection. Only the people with a valid password can access the file or folder. A file or folder can have more than one set of passwords which are used to identify the permission setting.
- If a file or folder is shared to guest, then it is considered to be published. Administrator can disable setting this permission for certain user group.

3.3. Non Functional Requirement

- **Performance/Time:**
  Processing Time: The document processing time taken should be within 5 seconds. Since

document processing unit can be various, the processing time is different. A progress bar will be shown for the document processing.

Response time: This system need a much quicker response time than typical web application. As simulation of traditional web processor is required, user needs an instant response like offline system.

- **Usability:**

  Usability is a crucial point in the system. As most users don't have experience of system like this, users are expecting to use the system in a way like a traditional word processor. The system should simulate the traditional operating system and word processor.

- **Security**

  All input need to be encoded and validated to prevent SQL injection. User can only perform operation under the permission. Some user groups can be configured that they can never have certain permission. For example, a user group can be set to only have read permission, namely guest.

- **Portability**

  The system needs to be portable on all major platforms. This system should not be restricted by any specific technology such as database, web server, and operating system. There should always be alternative environment.

- **Open Source**

  This system needs to be open source. As new document types are emerging and every organization has their requirement for this system, Most of the document processing units should be developed from different project. As a result, this system needs the power from open community.

3.4. Technology Specification

What technologies make this project possible? To put it simple, Web makes it all possible, and to take one step further, HTML makes the proposed system possible. In the proposed system, as can be seen from figure 2.3, the document will be encoded in HTML most of the time. The only exception is when the user needs the document of the original format.

The process of working with a document is that when the user chooses to open a document, the server returns the HTML version of the original document to browser. User finishes editing the document, and save it back to server. The server does not apply the change to the original document. Next time, when the user edits this document, the html version of last time is returned to browser without the conversion process. The change only will be applied to original document when user chooses to download the original type of document.

In one word, ***all jobs the system is performing are for the conversion between HTML and the Original Document Type.***

To implement this idea, the system will be built on the basis of C#, JavaScript, and MySQL. The reasons are stated below. Before analyzing any detail of the technology, the operating environment is investigated first.

3.4.1. Operating Environment

The system has a preferred environment: Windows 2003, IIS 6.0, MySQL 4.0, Microsoft .net

Framework 1.1

In fact, this system can run on Linux, Solaris, Mac OS X, Windows, and Unix with Mono framework. Additionally, the database system can possibly be replaced by Microsoft SQL Server or PostgreSQL. The web server can be replaced with Apache. These alternatives will be stated in following content.

3.4.2. Client Side Technology

On the client side, JavaScript and html are the choices. VBscript and PerlScript are candidates but they are not available on every browser. VBScript is only supported by Microsoft Internet Explorer, and Perl Script is only supported by installing plug-in to browser. XML can be a candidate for HTML, but in our application, most XML's features will not be utilized. Since the HTML editor is used on client side, complexity will be added if XML is used instead of HTML. However, a technology called Ajax will make use of XML to communicate between client and server.

Ajax is short for "Asynchronous JavaScript language and XML". It is not particularly a new technology. It is a combination of XHTML, XML, XSLT, XMLHttpRequest Object, Document Object Model and JavaScript. This technology has been available from 1998 when Microsoft created XMLHttpRequest in Internet Explorer 5.0. However, it is overlooked by developers until the recent high-profile implementation on Gmail, Google Map, and Google Suggest. This technology revolute the way web page works. Ajax technology enables JavaScript on the client side to make requests to the server without reloading the page. That makes it possible to develop more fluid and responsive web applications. The underlying idea is to make use of object "XMLHttpRequest" built in the browser. JavaScript firstly make a request to the server, and the server process the request then return the requested information in XML format. The XML information is parsed on the client side, finally JavaScript change the page on the client side according to the information.

3.4.3. Server Side Technology

On the server side, many technologies are available. PHP, JSP, ASP, ASP.net, Perl, Python etc. are all candidates. For the consideration of the scale and nature of this project, only JSP and ASP.net are candidates. Scripting language like Perl, Python are not well supported by all word processors, which is not suitable for the nature of this project. For the development of this system, interaction with Microsoft Word is needed. Java is not naturally supported by Microsoft Word, but with Asp.net and C#, Visual Studio Tool for Office 2005 can be used, which provides ready support for Office product. Additionally, .net is also supported by OpenOffice API. Java is well-known as cross-platform. However, as Bjarne Stroustrup indicated (Bjarne Stroustrup, 2006): it is not even close. It only runs on its own platform (JVM). The JVM platform is like Windows which is a proprietary commercial platform. This platform is owned by corporation and tweaked for the commercial benefit of that corporation. If even Java means "Cross-Platform", C# means much more if it is used with Mono platform. The Mono-project is an open source project which makes C# available on Linux, Solaris, Mac OS X, Windows, and Unix. Moreover, mono platform enables C# classes to work with Java class, which will make this system to support both Java and C# document processing unit. Since now C# can also cross platform, and it is widely supported by

word processors, it will be employed rather than Java. Although cross platform are mentioned several times here, Windows 2003 will still be the preferred environment. The code will firstly be compiled under Microsoft .net Framework and then switch to Mono .net Framework. As on Unix/Linux, Microsoft Word cannot be used, the server must compromise to use OpenOffice Writer to handle Microsoft Word document type.

3.4.4. Database

For the database, instead of sticking to one, a database abstraction layer will be used. This can ensure the system run cross different databases. At the time of writing, an open source software "Neo" has been chosen, as it supports some major database systems. According to Neo official website (Neo, 2004), one of reasons of using is "You deal with objects and objects only. All queries are expressed in terms of classes and properties with Neo creating the required SQL statements at runtime." This is the essential point for all object persistence frameworks like Neo. (CSharp-Source, 2005) Developer only focus on the object oriented implementation, and the detail of database programming can be generalized in an object oriented way.

3.4.5. Open Source

Open source is not omnipotent, and it has lots of limitations. The project is not blindly pursuing open source. Open source suit the nature of this project.

The existing document type is various, and new document type is emerging. Hence, every people has special requirement to this system. For example, organizations in China might need support for WPS file, whereas people from other countries do not need it. Moreover, there are still a wide variety of extension needed, such as the image processing and multi-language support etc. The final target of this system is to cover all human requirements related to document processing. The only way for the author to achieve the target is to make this system open source, and more people can join the development of the system. OpenOffice is a good example of how open source can help document processing system.

3.5. User Case Diagram

Use case contains:

3.5.1. User System Diagram

3.5.2. File System Diagram

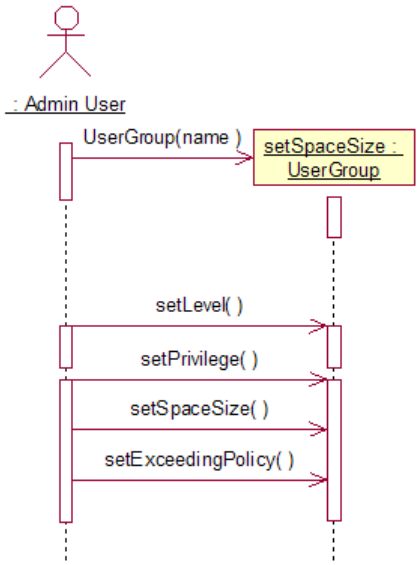
4. Design

4.1. Class Diagram

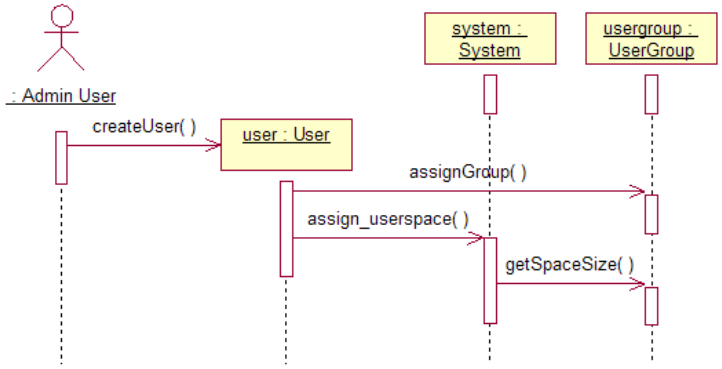
The class diagram is included in the appendix 2. Class diagram.

4.2. Sequence Diagram

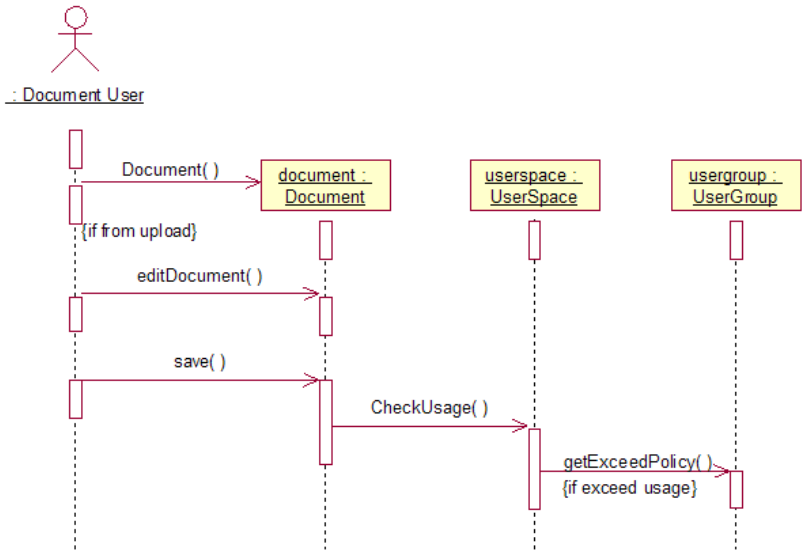
4.2.1. Create User group



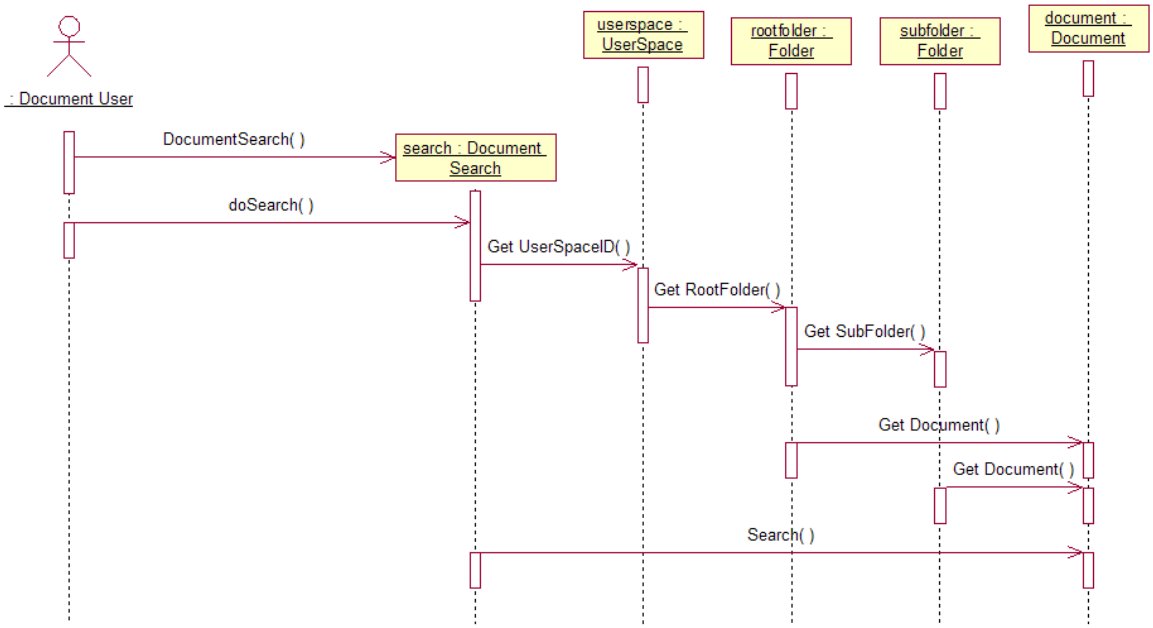
4.2.2. Create User



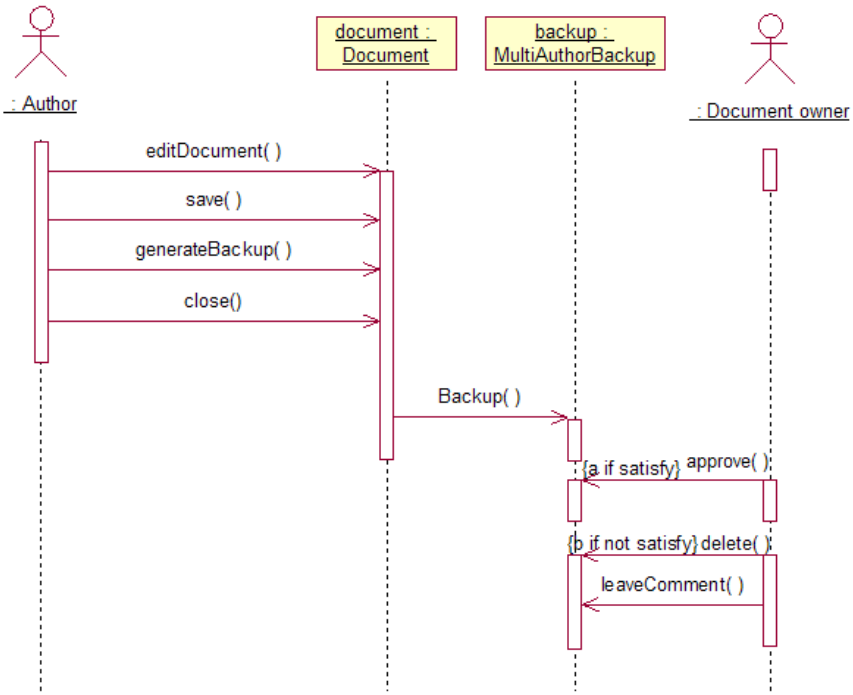
4.2.3. Create Document



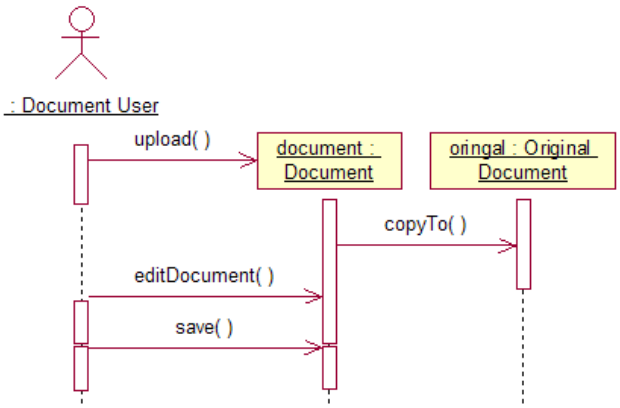
4.2.4. Document Search



4.2.5. Multi Authoring



4.2.6. Open Local Document



4.3. Database Design

The database design is directly mapped from the class diagram. The path attribute for File and Folder table is optional, because the path of a file can be calculated by the relation between parent folders. The Entity relational diagram is included in the appendix.

5. Implementation

In the implementation section, firstly the difficulties and the reasons leading to a specific approach are explained. Then, the basic components of the system like Window and Document processing unit are stated. The other aspects of the system are explained briefly at the end.

The difficulties of the implementation are numerous. The fundamental difficulties are firstly from the shortage of debugging tool. It is hard to debug JavaScript and even harder to debug Ajax. The Visual Studio is not designed for Ajax, or it is better to say that there is no serious debug tool for Ajax and ASP.NET at the time of writing. In order to have an intuitive feeling of the JavaScript code, the components such as window are isolated as a single html file. The file is then read into the main page at runtime through Ajax.

The second difficulty is the some initiating work of the web operating system in this project. Despite the term “WebOS” is out for years, there is still not a non-trivial “window” control. (Imagine an operating system without window!) It could be very hard to develop c# desktop application without a window form control. It is same for the rich internet application which declared to blur the distinction between desktop application and web application. Hence, this project needs to start completely from scratch. (Imagine writing windows application that windows control must be developed first!) Not only is the window control, but also the right-click context menu is the similar case.

Inter-Browser Drag and Drop Window

The “window” control is one of the most critical yet fundamental part of the project. It enables real Web operating system applications. The technologies involved are CSS, JavaScript and HTML.

The basic idea behind the windows simulation involves dynamically adding content from server side at runtime to the client page asynchronously. It is obvious unacceptable for any synchronous approaches, because user tends to do multiple tasks concurrently, for any synchronous approach if a user open a window, the client has to synchronize with server. In the meantime, user can do nothing. Furthermore, in order to maintain the window states within the browser,, the server side has to store complete and precise window state details every time user make requests to the server and apply the stored states back to the client in the server response. It is very complex and error prone process.

Basic structure

The source code of basic components of a window template are included the the file “WebForm1.aspx”.

The basic process of generating window is designed as follows:

Firstly, when the page is loading a window template is cached into the client.

```
var baseWindow=FileSystem.Index.newWindow("WebForm1.aspx");
```

Secondly, Every time the new window is opened, a new window is initiated from the existing window template. The generate source code is appended onto the end of the page.

```
function addWindow(windowName,title){
var out=setProperty(baseWindow.value,"w",windowName); //Specify the window ID
out=setProperty(out,"Title",title); // Specify the title
document.write(out); // Append to the end of the document
```

```
windowArray.push(windowName); // Add new process to the process list.
arrangeWindows(); // The function which arrange all windows
eval("dd.elements."+windowName+"titlebar.maximizeZ()"); //Bring the window to front.
}
```

This structure can significantly reduce the server load. Since the only server transaction is at the initiation of the page, once the page is loaded, any new windows are all initiated from client with no server side processing. ("Window" mentioned here means the window frame itself but not any content in the window)

HTML Template System

As mentioned,"window template" is loaded on the page initiation. Any new window needs to be initiated at the client through Javascript. The initiation is done trough "setProperty" function. This function substitutes the template anchor in the window template with the actual property value. In other word, another rendering layer besides ASP.Net is added on the client side. Hence, the "window template" here means the window source code combined with template anchor that to be substituted with the actual content. The system is like any tag systems such as ASP.NET and PHP. In asp.net, tag "<% %>" is used. In this system, tag "* *" is used.

For example, every window should have a unique name to identify the window and its sub components. Therefore, a template tag "*w*" which stands for window name is to be substituted with the actual window name. In order to set the name of the window namely substitute *w* with the actual window name, the setProperty(baseWindow, property, value) method is used as following:

```
var out=setProperty(baseWindow.value,"w",windowName);
```

Variable "out" contains the modified version of window template, which can then be appened to the end of the document and the window is displayed.

After introducing the general process of generating a window, the basic sub-components of a window are explained below.

Basic components:

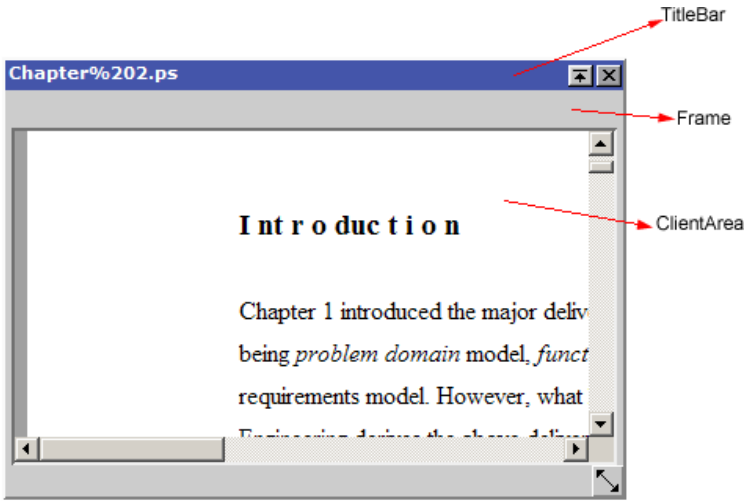


Figure 1 Basic Window Components

For a simulation of a window, the window control contains threes components: "Titlebar", "Frame

“and “Clientarea”. These components are all based on HTML Division tags, and their styles are controlled CSS and dynamically controlled by JavaScript.

Title bar: Display title and window control buttons.

Frame: the function of “Frame” is to display the border and frame to simulate the style a window.

Client Area: the function of client area is where the actual content of the window is located. There are two types of client area, one is used to display small amount of messages, all the content is written directly into the division tag. For the second one, the division tag can include a complete page via the <IFrame> tag. This can be used to display a large amount of information.

Others: There are other components like the resize handle, the close and minimization button.

The HTML source code of the window components without any assembling is included in appendix 1.1.

Assembling components

The purpose of this process is to assemble all the components above together to make them works as a whole. The size and position of the each component are controlled by JavaScript. Five events of the window are identified, which are:

*(*w* would be substituted by the window name on initialization)*

Event name	Condition	Description
*w*initWindow	window is loaded	Initializes the components, and assemble them together
*w*my_PickFunc	when a window item is hit by a mousedown event	Handles any component is clicked including when the close button or the minimization is clicked, and when a window is selected.
*w*resizeWindow	invoked while the window is resized	Resize the components according to the coordination of the resize handle.
*w*my_DragFunc	when the window is dragged	Relocate the components according to the coordination
*w*my_DropFunc	when an item is dropped into the window.	

The implementation detail of the function “*w*initWindow” which assemble all components is shown below.

```
function *w*initWindow()
{
dd.elements.*w*titlebar.moveTo(dd.elements.*w*frame.x+2+*w*frame_padding,dd.elements.*w*fram
e.y+2+*w*frame_padding); // Move the title bar

dd.elements.*w*titlebar.addChild("*w*frame");
dd.elements.*w*titlebar.setZ(dd.elements.*w*frame.z+1); // ensure that *w*titlebar is floating
above *w*frame

dd.elements.*w*titlebar.resizeTo(dd.elements.*w*frame.w-4-(*w*frame_padding<<1),*w*titlebar_
h);//Resize the title bar

dd.elements.*w*clientarea.moveTo(dd.elements.*w*frame.x+2+*w*frame_padding+*w*clientarea_mar
gin, dd.elements.*w*titlebar.y+*w*titlebar_h+*w*toolbar_h+*w*clientarea_margin); //Move the
clientarea

dd.elements.*w*titlebar.addChild("*w*clientarea");

dd.elements.*w*resizehandle.moveTo(dd.elements.*w*frame.x+dd.elements.*w*frame.w-dd.elements
.*w*resizehandle.w-2,dd.elements.*w*frame.y+dd.elements.*w*frame.h-dd.elements.*w*resizehand
le.h+1);//Move the resize handle

dd.elements.*w*resizebutton.moveTo(dd.elements.*w*titlebar.x+dd.elements.*w*titlebar.w-dd.el
ements.*w*resizebutton.w-*w*frame_padding-(*w*titlebar_h>>1)+Math.round(dd.elements.*w*resiz
ebutton.w/2)-18,dd.elements.*w*titlebar.y+Math.round(*w*titlebar_h/2)-Math.round(dd.elements
.*w*resizebutton.h/2));//Move the resize handle
```

```
dd.elements.*w*closebutton.moveTo(dd.elements.*w*titlebar.x+dd.elements.*w*titlebar.w-dd.elements.*w*resizebutton.w-*w*frame_padding-(*w*titlebar_h>>1)+Math.round(dd.elements.*w*resizebutton.w/2),dd.elements.*w*titlebar.y+Math.round(*w*titlebar_h/2)-Math.round(dd.elements.*w*resizebutton.h/2)); //Move the close button

dd.elements.*w*titlebar.addChild("*w*resizebutton");//Add minimization button to titlebar
dd.elements.*w*titlebar.addChild("*w*closebutton");//Add close button to titlebar
dd.elements.*w*titlebar.addChild("*w*resizehandle");//Add resize handle to titlebar
dd.elements.*w*titlebar.setPickFunc(*w*my_PickFunc); //Assign my_PickFunc to the "pick" event of the titlebar

dd.elements.*w*resizebutton.setPickFunc(*w*my_PickFunc);
dd.elements.*w*closebutton.setPickFunc(*w*my_PickFunc);
dd.elements.*w*resizehandle.setDragFunc(*w*my_DragFunc);
dd.elements.*w*titlebar.setResizeFunc(*w*resizeWindow);
dd.elements.*w*titlebar.setDropFunc(*w*my_DropFunc);
dd.elements.*w*resizebutton.setDropFunc(*w*my_DropFunc);

dd.elements.*w*titlebar.hide(); //Hide the whole window, latter the window will be shown

}
```

The full implementation details of the above functions are shown in the appendix. After assembling all the components, the basis of a single window is completed. Next, the issue of multiple window arrangement is concerned.

Multiple windows arrangement

To arrange all the windows, the windows need to be stored into an array. This array can be regarded as the "process list" or "task list" of the mini web operating system.

The only way to arrange multiple "window" appearance in DHTML is to use the CSS property "z-index". The larger the z value of the element is the upper layer the element is on. Rather than bringing the window forward and backward, for the simplification the window is designed to be forward only in the actual implementation of this project. Method "maximizeZ()" is created to lift the window to the currently highest z-index.

The implementation detail of the method is included in the appendix. The basic idea is to examine the z-index of all windows in the current process list, and generate a larger z-index value and apply it on the current window.

Display Functions

By far, the window control is available. The content can now be loaded into the window. There are a series "Display" function that load various types of content into the window.

Function Name	Usage	Implementation Detail
displayImage	Display image	Write a <img> tag into the client area and load the url of the image into the src property.
displayLoginWindow	Display the login window	
displayPDF	Display PDF file	Write a <iframe> tag into the client area and load the processed document into the window
displayPS	Display PostScript file	Same as above
displayDOC	Display word file with editor	Same as above
displayReadOnlyDoc	Display word file in read only mode	Same as above
displayText	Display plain text file	Read the text from server side and add it to the client area's "innerHTML" property

Document processing

By far, the display of various type of document is completed. Now how the document is processed is focused. Developing processing module is beyond the scope of this project. The author's effort is on the integration of the existing modules into the system. The various processing modules involved in the system are listed below

Processing Module	Description	Author's effort
PDF2HTML	The module is coded in C, which simply convert any PDF file to HTML. It is based on GSViewer and xpdf 2.02 by Derek Noonburg.	The author's effort mainly involve in config the server side to make it work with ASP.net.
PS2PDF	The module comes with GSViewer and it is a batch script that converts PS file to PDF file.	Similar works as in PDF2HTML and additional work to make it work with PDF2HTML which subsequently convert PDF file generated to HTML.
Microsoft Word	Microsoft word comes with Marco languages which can perform any operation programmatically.	Use the Marco language to operate the Microsoft word programmatically

The excerpt of c# code that converts the Microsoft word to the HTML file and plain text file is shown below(the complex parameters are represented by "…");

```
Word.ApplicationClass oWordApp = new Word.ApplicationClass();//Create Word application instance
oWordApp.Visible = false;//Invisible on serverside
object missing = System.Reflection.Missing.Value;
object format = Word.WdSaveFormat.wdFormatDocument;
object format2 = Word.WdSaveFormat.wdFormatFilteredHTML;
object format3 = Word.WdSaveFormat.wdFormatText;
Word.Document oWordDoc = oWordApp.Documents.Open(...);
oWordDoc.Activate();
oWordDoc.SaveAs(...);//Save the document as HTML format which is format2;
oWordDoc.SaveAs(...); //Save the document as PlainText which is format3;
oWordDoc.Close(...);
oWordApp.Application.Quit(ref negative, ref missing, ref missing);//Quit the application
```

There are other aspects of implementation of the system that will not be investigated in detail in this report. It includes various file operations, rich text formatting. These aspects will be described briefly below:

File Operations:

The file operations such as "remove" generally involve three steps:
Firstly, the file is selected by the user. The server side path of the selected file is stored on the "currentSelection" variable on the client side. Secondly, when the user clicks any operation button like "remove", the command with the path of the selected file is sent to the server side by Ajax. Thirdly, the server side performs the command such as "remove", and return the result to the client side. The client side perform the refresh operation to retrieve the new file list.

Rich text formatting:

These operations are used to simulate any rich text editor like Microsoft Word. The typical operations include bold, font style and so on. The majority of these operations are all based on the "**iframe.document.execCommand()**". The general scenario of these operations is in three steps. Firstly, the user selects an element like some text. The selected element can then be referenced

from some built-in variable like “**document.selection.createRange()**” (This command sometimes is optional). Secondly, the user clicks the formatting button like “bold”. Thirdly, the **iframe.document.execCommand()** executes the corresponding command. For example: **document.execCommand('Underline').** This command simply add underline style to the selected text

6. Evaluation and Test

6.1. Internal test

6.1.1. Document Test

The basic test plan is to firstly open the document in its original editor, and then open it in the web based editor. Finally, the results are compared. To prove the document is actually opened in our web system but not in the original reader, the content of the task manager is monitored and shown with every figure.

6.1.2. Portable Document Format(PDF)

In Figure 1, a complex PDF document that contains vector graphs and texts is displayed the Acrobat Reader 7.0.

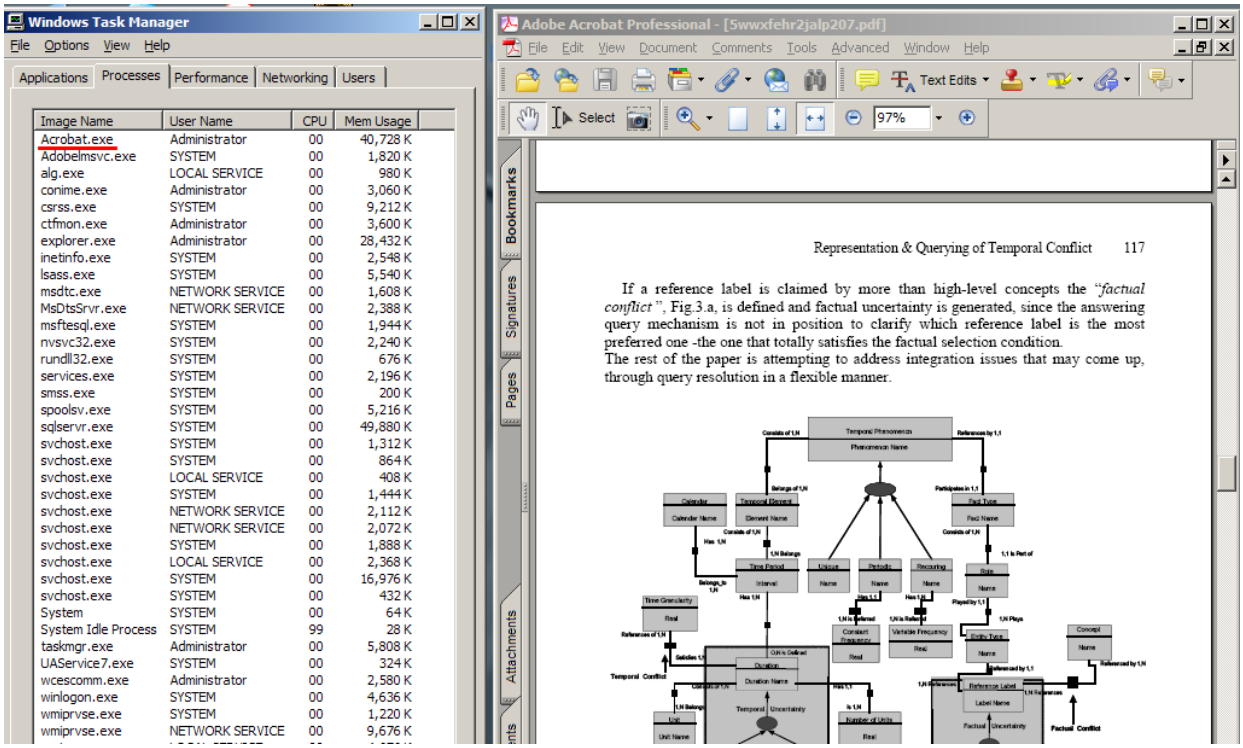


Figure 2 when it is opened in Acrobat Reader 7.0 Professional:

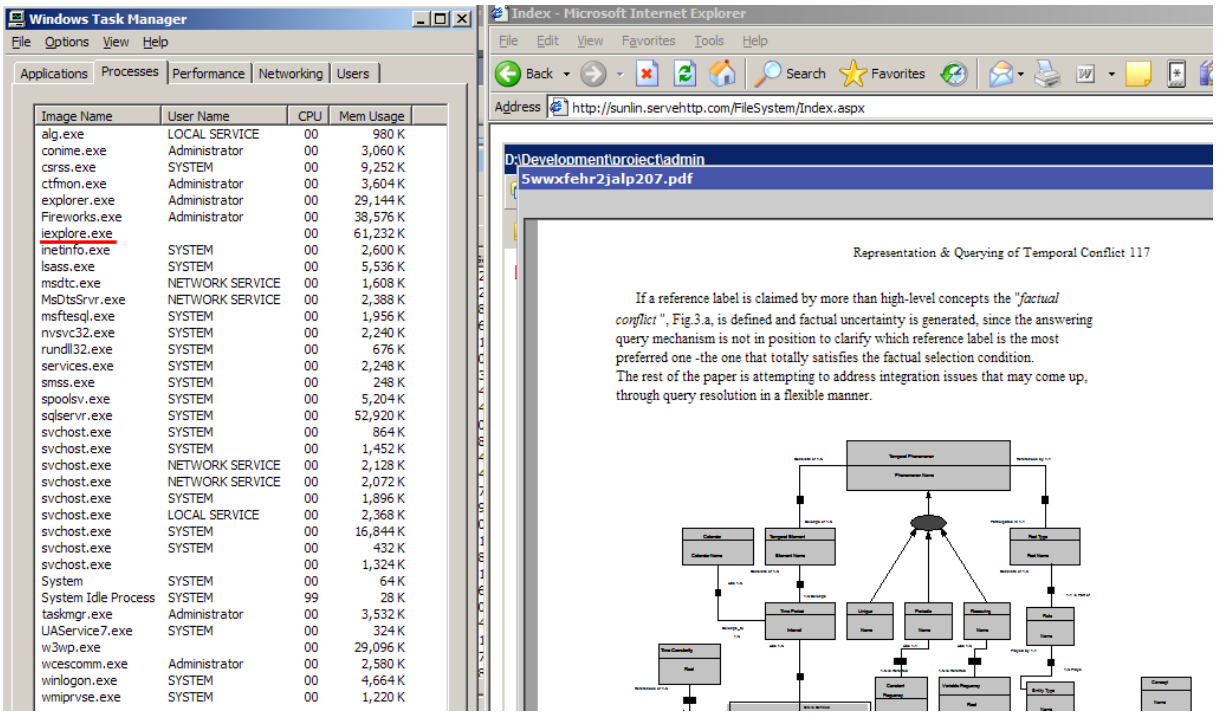


Figure 3 The PDF document is opened in the web based editor.

Result: The vector images and text are displayed the same as in the Acrobat Reader. The text on the vector image is still selectable. In conclusion, the PDF file can be displayed with in the browser and without the support from Acrobat Reader.

6.1.3. Postscript

In Figure 3, a complex postscript document that contains vector graphs and texts is displayed the GSView32 version 4.7.

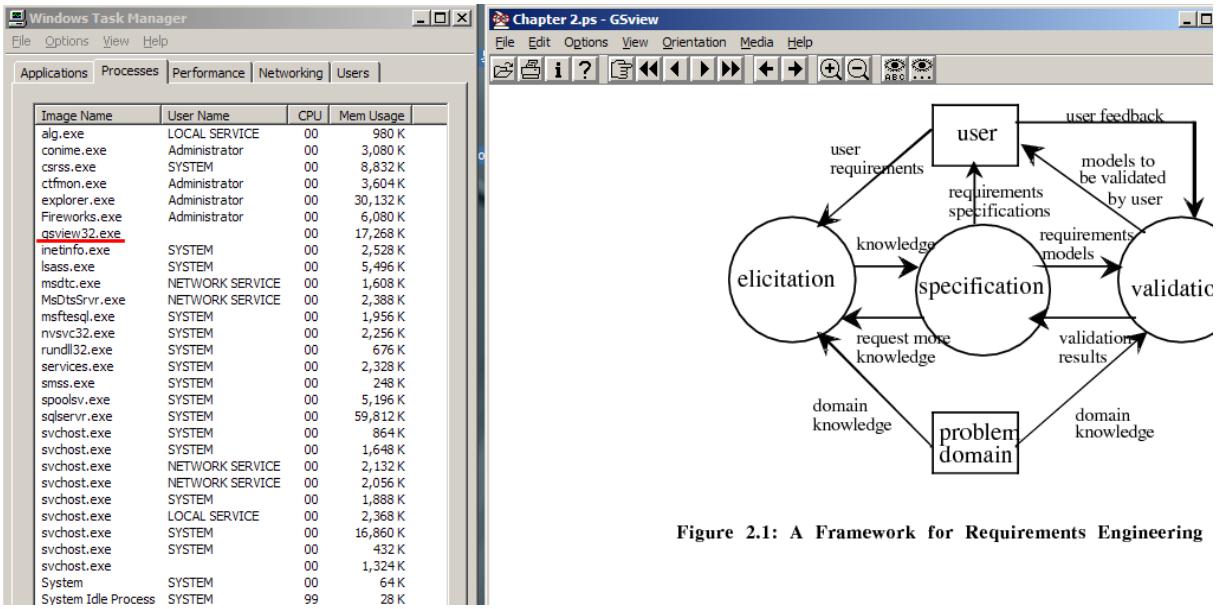


Figure 4 A postscript document is opened in GSView32 version 4.7

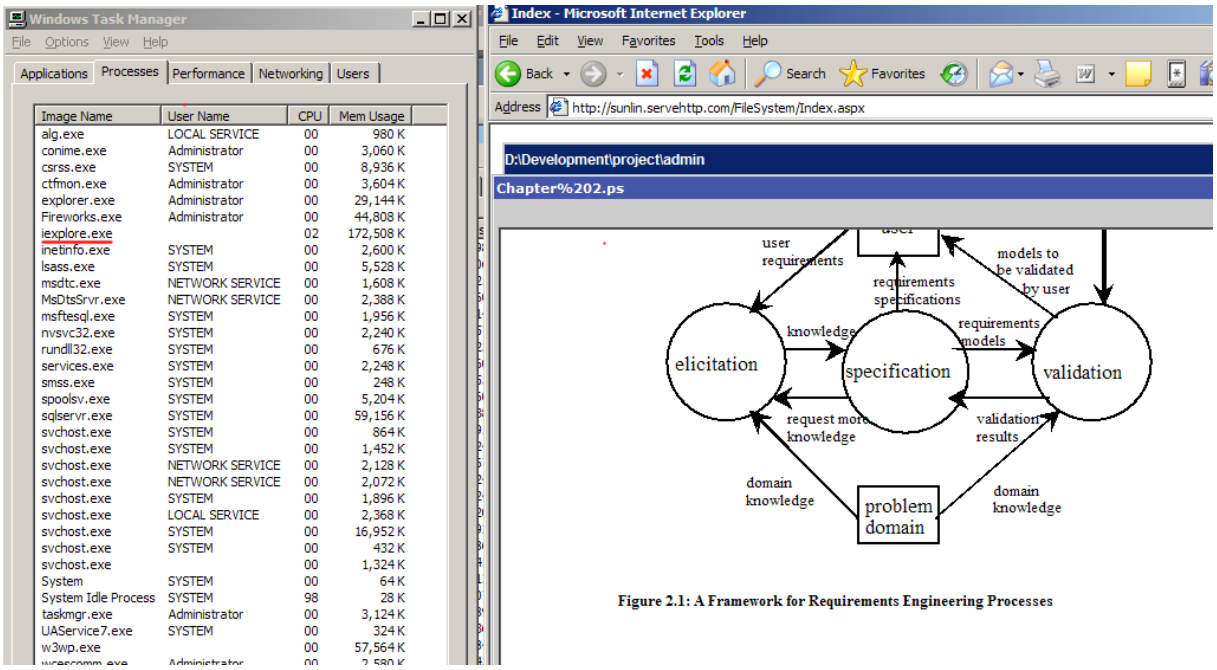


Figure 5 a postscript document is opened in the web based editor.

Result: The vector images and text are displayed the same as in the GSView32. The text on the vector image is still selectable. In conclusion, the postscript document can be displayed with in the browser and without the support from GSView32.

6.1.4. Microsoft Word Document

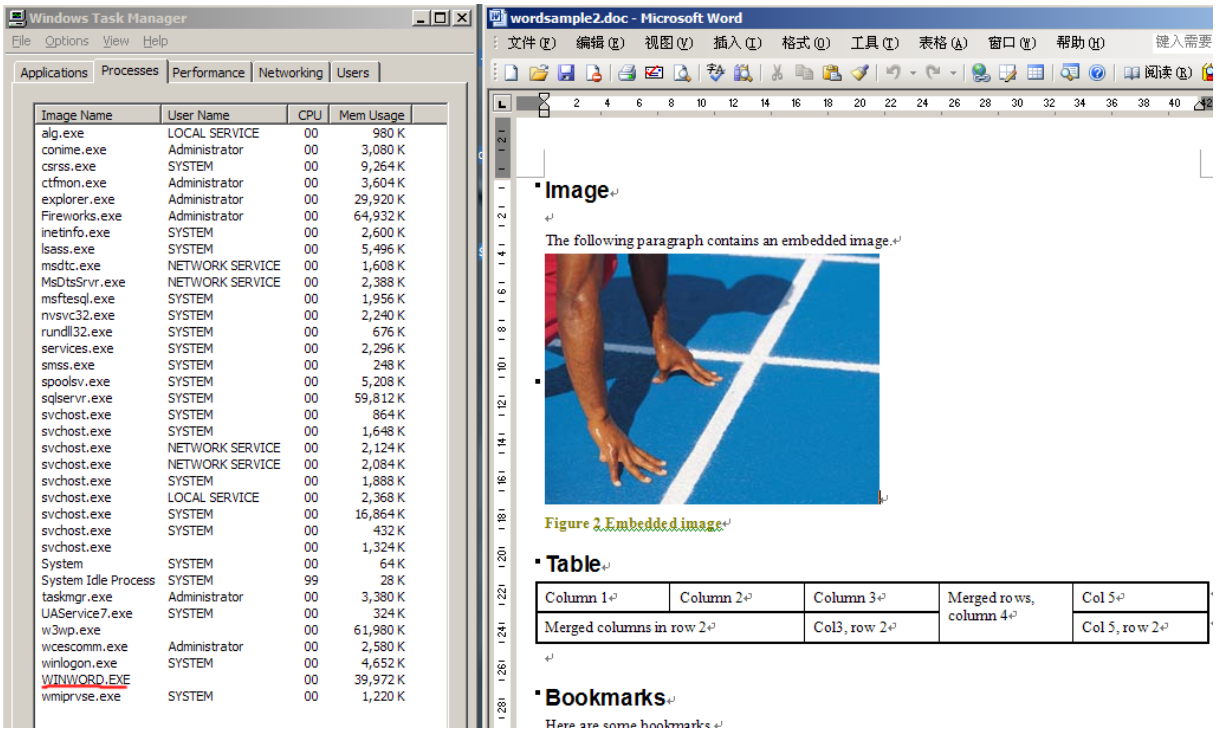


Figure 6 A word document is opened in Microsoft Word 2003

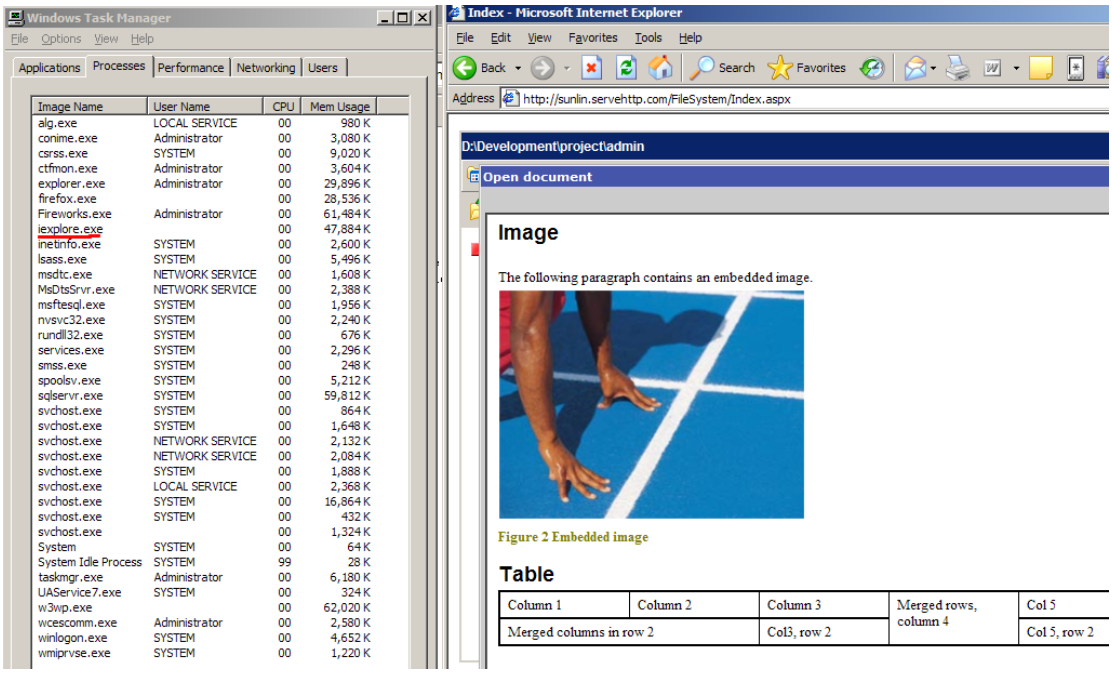


Figure 7 a word document is opened in the web based editor.

Result: The images and text are displayed the same as in the Microsoft Word. The various text styles and tables keep its original format in the web based editor. In conclusion, the Microsoft word document can be displayed within the browser without the support from Microsoft Word.

6.1.5. User interface test:

6.1.5.1. Window test:

As a feature of the system, a “Window” control is created based on DHTML and JavaScript, and it is one of the most complex parts of the project. As it is a totally DHTML+Javascript widget, the compatibility on different browsers is very important. The following tests are performed both on Internet Explorer 6.0 and FireFox 1.5.

Resize window

Test plan: Open a window, and use the resize handle to increase and decrease the size of the window.

Result: The window can be resized using the resize handle. The windows resize function works as expected even in extremely large or small size.

Figure 8 before the window is resized

Figure 9 after the window is resized

Windows arrangement

Test Plan: Open four windows and rearrange the windows.

Result: The windows works the same as in Microsoft Windows operating system except a unsolved problem as indicated below.

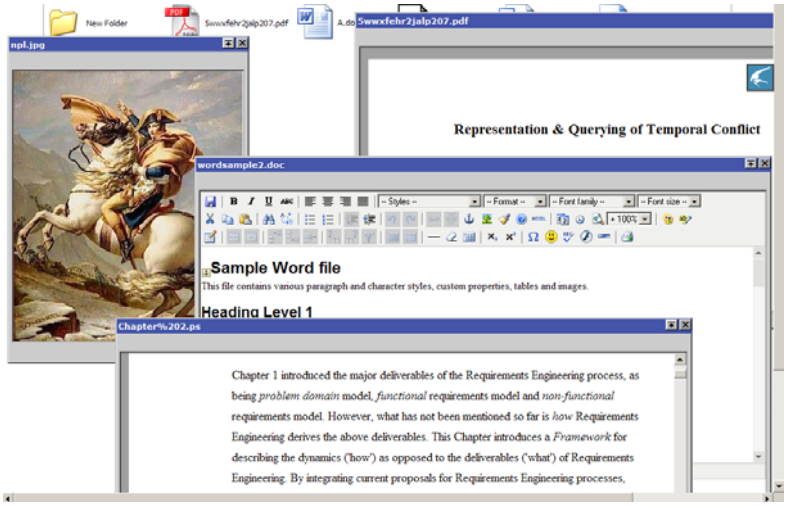


Figure 10 Illustration of multiple windows

There is an unsolved problem when multiple windows are arranged on the Internet Explorer 5.5 and above.

Figure 11 Unsolved problem of arrangement

As shown in the picture above, the select list is always drawn on top of image and iframe. The select element is rendered as “windowed element”. The Iframe element was rendered as “windowed element” before Internet Explorer 5.5, but is rendered as “windowless element” in Internet Explorer 5.5 and beyond. All windowed elements paint themselves on top of all windowless elements, despite the wishes of their container. In conclusion, the select list will always be drawn on top of any IFrame in Internet Explorer 5.5 and above. Solving this problem is beyond the scope of this project. It cannot be solved unless new version of Internet Explorer interpret select element as a “windowless” element as in Mozilla.

Minimize window

Test Plan: Open a window, then click the minimization button, finally click the button again to return the size.

Result: The windows can be minimized and after that the size can be restored.

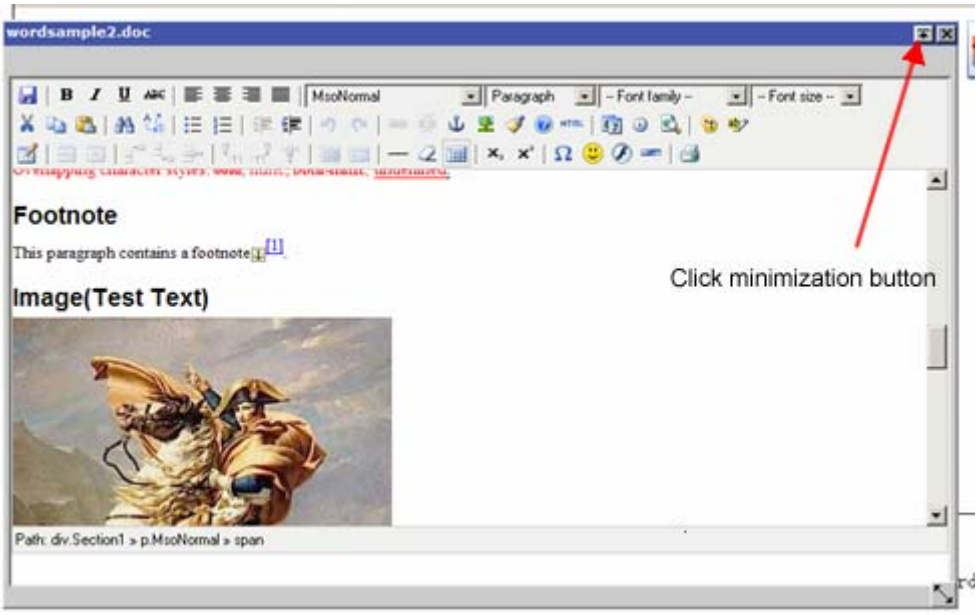


Figure 12 before minimization

Figure 13 after minimization

Close window

Test Plan: Open a window, and click the close button to close it.

Result: The original window is shown as in Figure 12. When the close button is clicked, the window is closed.

6.1.5.2. File Explorer

The typical file system operations are tested, and it all works like in a typical operating system. These operations includes “Cut”, ”Copy”, ”Paste”, ”Create File”, ”Create Folder”, ”Delete File”, ”Delete Folder”, “Rename”. For the length of the report, only the test of creating folder is shown as an example.

Firstly, the create folder button is clicked.

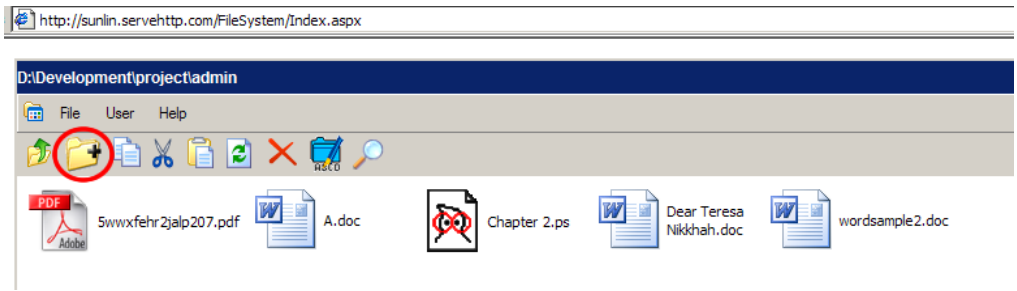


Figure 14 the location where the mouse is clicked

Secondly, the system prompts for the folder name.

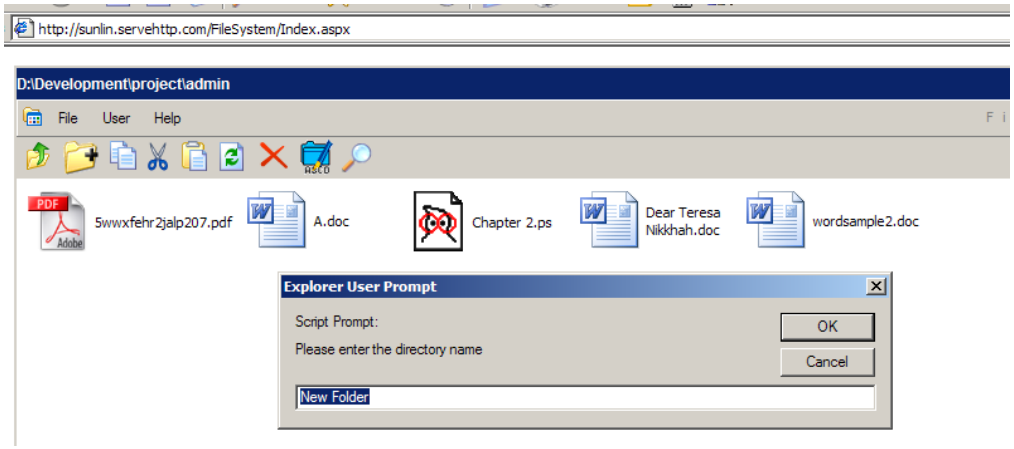


Figure 15 the folder name is supplied

Thirdly, when the OK button is clicked, a folder with the specified folder name is created

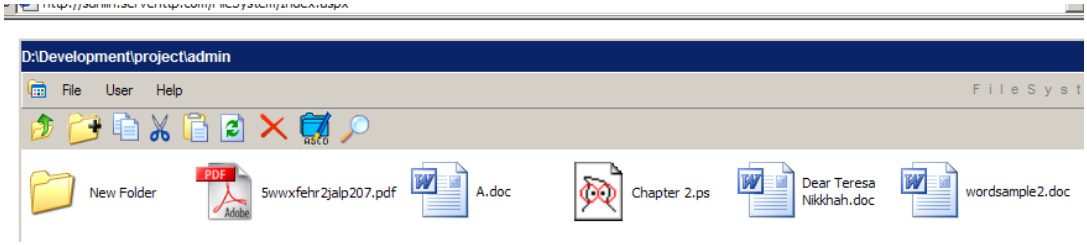


Figure 16 the folder is created

6.1.5.3. Word Editor:

The word Editor contains almost all functions of a typical rich text editor, including functions of font manipulation, image manipulation, and Table manipulation. All functions are tested, but for the length of this report, the test for font manipulation is shown here as an example.

First, string “this is a test string” is typed into the editor.

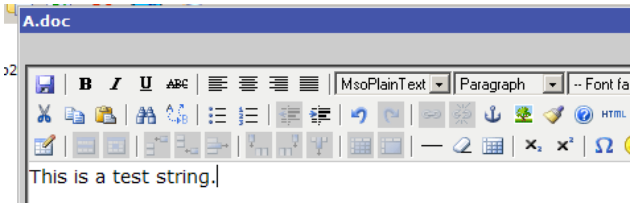


Figure 17 text before bold

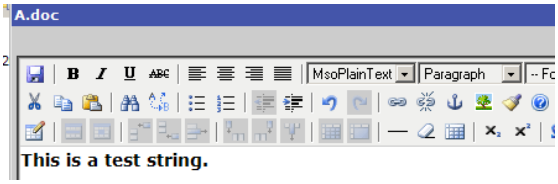


Figure 18 after “bold” operation:

Figure 19 change the font size to “36pt”

From above test, it can be proved that the different styles can be applied to the text. Additionally, the interaction is based on WYSWYG.

6.2. Integration Test

Test objective:

This test checks if the changes to the document take effect and the compatibility with the original editor. The tests are done for 20 mostly used functions. Image insertion and text insertion is shown below as an example.

Test Plan:

From the web based editor, some text will added into a sample document and an image will be inserted into this sample document as well. Then, the document is downloaded and opened with Microsoft Word to check if the change takes effect.

Test process and result for each step:

The document is in Microsoft word format, and its content is shown below. How the document looks like in Microsoft word editor is shown in Figure 5.
Firstly the text “Test Text” is added, and then the picture is replaced with another picture.

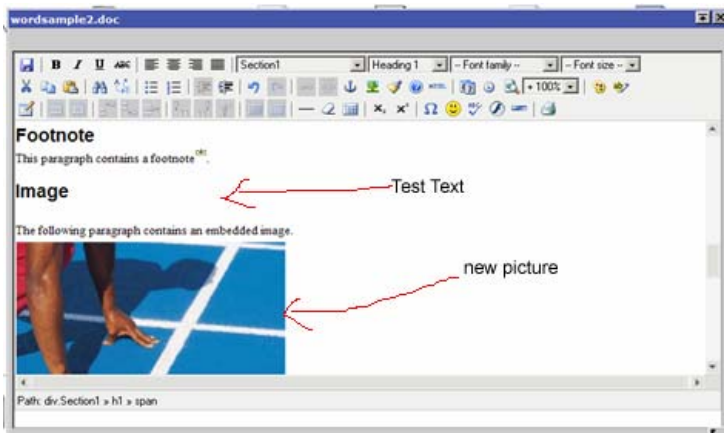


Figure 20 The part of the document need to be changed

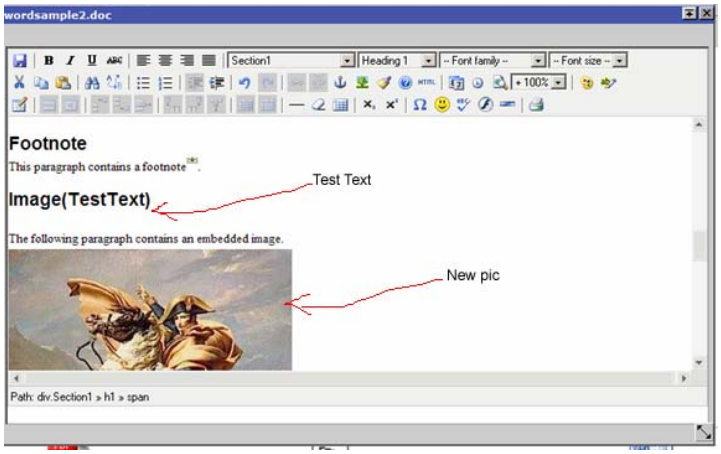


Figure 21 after change

Secondly, the save button is clicked. The change is saved, and the document is then opened in Microsoft word to check if the changes take effect in the original document.

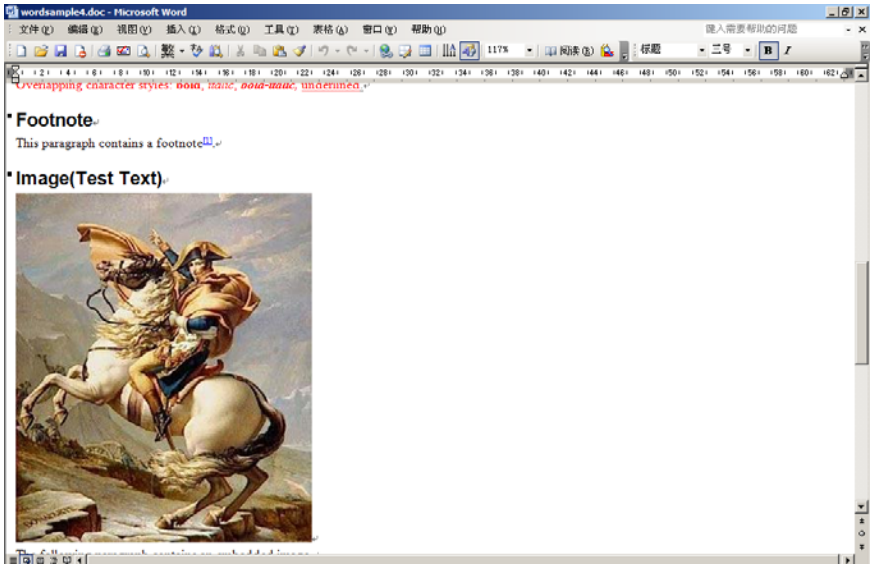


Figure 22 The changed document is opened with Microsoft Word 2003.

As shown above, the changes take effect in the original document. And it also can be observed that the rest part of the document keeps its original format. In conclusion, after various tests, it can be proved that the word document edited by the web based system can be fully integrated with Microsoft Word.

File Sharing and Search

The file sharing and file search are tested together, because user can only access sharing function via searching.

Test Plan: Firstly, the user named “linlin” in the user group “User” log in and search the file available to him. (Figure 23)

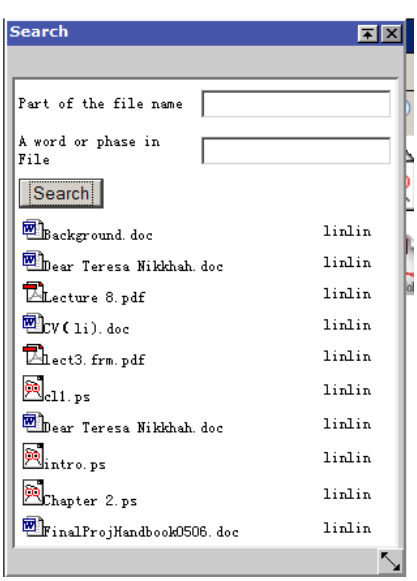


Figure 23 documents available to use “linlin”

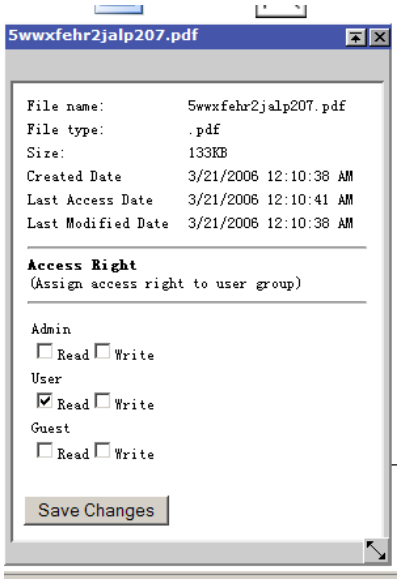


Figure 24 user “admin” share a f the user group “User”

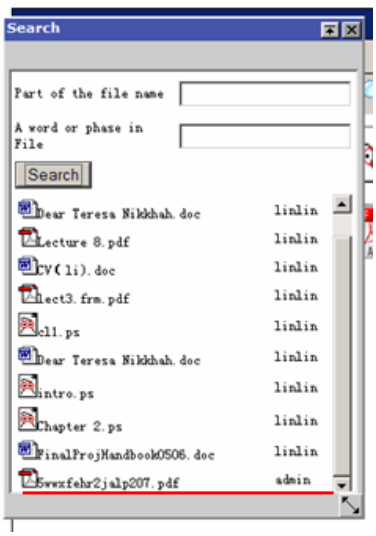


Figure 25 documents available tc “linlin”

Secondly, the user named “admin” who is in “Admin” user group shares a file to the user group

“User”. (Figure 24)

Thirdly, the user named “linlin” in the user group “User” log in and search the file available to him again. The file listed in Figure 25 is compared with the file listed in figure 23 to inspect the difference. The difference is that the file “5wwfehr2jalp207.pdf” which is shared by user “admin” is available for user “linlin” now.

Finally, the user “linlin” open the shared file.

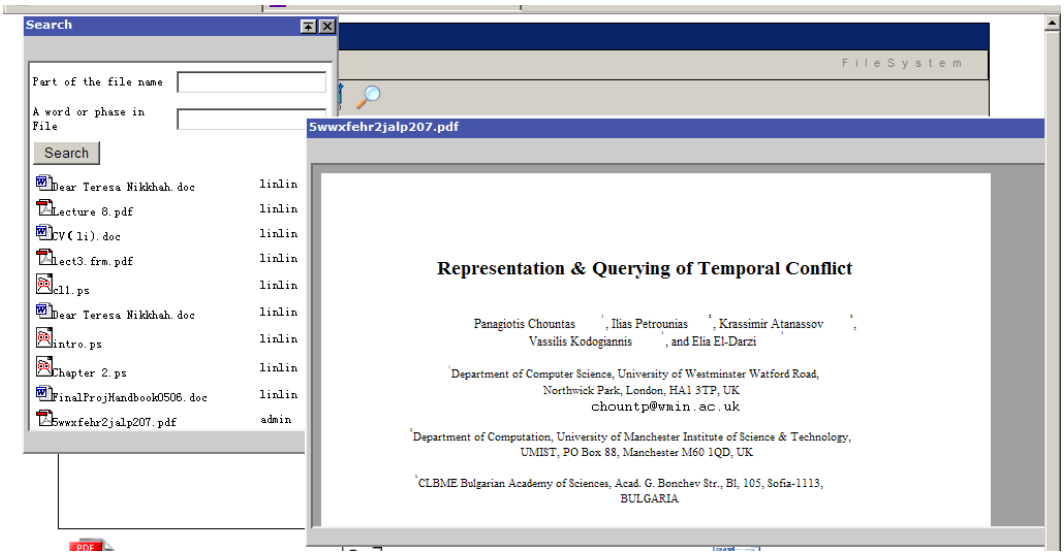


Figure 26 open the shared document

Test Result:

By repeating the tests a number of times, it can be proved that the user can access the file owned by him and shared to his user group.

6.3. External test

In this section, to what extent does the project satisfy the stated requirement is evaluated in this section.

6.3.1. Functional Requirement

Document: User can read PDF, PostScript and Microsoft Word document. User can edit the Microsoft Word document. The optional requirement of “Imaging a document” is not implemented yet.

User and User Group: All users are grouped but not the guest. The system follows a different logic for the guest from the designing stage. The guest is not grouped and will not be able to do anything. User and User group handling is implemented and fully implemented as stated so the user and user group can be manipulated as stated in the requirement section.

File and Folder: The manipulation of file and folder and implemented as stated. However, the file permission approach is not so elegant. A more valid approach should be integrated with the server side operating system. Because the operating system level is more secure than the application level.

6.3.2. Non-functional Requirement

Performance/Time: The requirements are mostly satisfied except for some complex Microsoft Word documents. If the word document's size is greater than 800KB, the processing time is typically more than 5 sec. A progress bar is shown to indicate the progress.

Usability: The usability of this system is tested by a number of people. The overall result is excellent despite user do need some time to familiarize with the system. The usability is not as good as traditional word processor. This system is after all a simulation of traditional word processor. User can perform basic tasks without learning the system. However, if user wants to perform all tasks, user might need to learn the system to certain depth.

User	Rated Result	Comment
Yu Zhang	Excellent	Works just like a desktop application. It could be improved if multiple selection of file is enabled.
Becky Lin	Excellent	Genius! But there is no alert for overwriting files, and the selection of file works slightly different from the windows operating system.
JiaZi Zhuang	Excellent	Excellent work. I do need some time to be familiar with the system. It is the first WebOS I have so far used. It could be helpful to add tips and animation to help user use the system.

Security: The security test for this system is satisfactory. The source code of the client is unchanged and only the necessary file is revealed. It is hard for any hacker to discover the server side structure to perform attack like SQL injection. Any file and user operations are checked for the requested permissions and privileges. The security model could be improved by integration with the server side operating system and Https.

Portability: The system is designed to run on both Mozilla and Internet Explorer web browsers, User won't spot any differences when migrate cross these browsers. Mozilla is available on a number of other operating systems like Linux and Unix. User on these operating system feel no difference to the Windows operating system.

In conclusion, from the internal and external test, the system achieves the functional and non-functional requirement. Except the security approach, the other approaches are among the most appreciate approaches from the author's viewpoint at the time of writing.

Some of the functionalities of the traditional word processor cannot be achieved, "Marco" and "Vector Image" are examples. The limitation of the web based editor is from the limitation of JavaScript and HTML, although there is some good attempt.(Walter, 2005).

7. Conclusion

In a conclusion of this report, the following aspects are discussed.

Chapter 1: the project is briefly introduced. In this section, the basic idea is stated which is used in the rest of report.
Chapter 2: the background of this project is described. The realization of Bill Gates' statement at 1994 namely 8A's principle serves as the basic motivation of the project. The comparison of on-market products further firm the resolution of developing a new web based system.
Chapter 3: the Requirements are analyzed. The functional/nonfunctional requirements are listed and prioritized.
Chapter 4: the design of the system is discussed in this section. The class and sequence diagrams are enclosed. The entity relational diagram is also included.
Chapter 5: The implementation detail of the system is revealed. The chapter starts with the implementation difficulties. These difficulties serve as a reason for particular approaches. Then the "Window" control and document process unit is examined in detail.
Chapter 6: The system is tested and evaluated. For every internal test, the test plans and test results are shown, and the results are discussed. The external test evaluates how the system satisfies the requirement specified in the requirement analysis section.

Skill Gained

The project starts from Sep 2005 and ends at April 2006. At the start of the project, the author has already been familiar with most web development technologies both on server and client side. By doing this project, the knowledge learned by the author is listed below.

New skill: Ajax Technology

Enhanced skill: ASP.Net with C#, SQL Server 2005, JavaScript, the formal approaches of software design and implementation, Requirement Analysis

In one word, the system developed from this project satisfies the requirements stated, and it is a very useful tool helping people to manage documents.

List of References

1. Alan, D., Barbara, H, W. & David, D. (2002) *Systems Analysis & Design: An Object-Oriented Approach with UML*, John Wiley& Sons, Inc.
2. Butler Lampson. (September 1979). Bravo Manual, *Alto User's Handbook*: Xerox PARC, pp 31-62)
3. Desk Top pipeline (2005). *Vote: Which word processor do you use?* [online]. Available from: http://www.desktoppipeline.com/vote/050131_word.jhtml **[Accessed 11/11/2005]**
4. Gsurface(2004). *AbiWord vs. MS Word* [online]. Available from: http://www.flexbeta.net/main/printarticle.php?id=78 **[Accessed 17/11/2005]**
5. James Robertson. (2003). *So, what is a content management system?* [online]. Step Two Designs. Available from: http://www.steptwo.com.au/papers/kmc_what/index.html **[Accessed 01/11/05]**
6. Mono(2005). *Supported Platforms* [online]. Available from: http://www.mono-project.com/Supported_Platforms **[Accessed 17/11/2005]**
7. Michael Arrington(2005) *Writely - Process Words with your Browser* [online]. Available from: http://www.techcrunch.com/2005/08/31/writely-process-words-with-your-browser/ **[Accessed 17/11/2005]**
8. Object Technology International. (February 2003). *Eclipse Platform Technical Overview* [online]. Available at: http://www.eclipse.org/whitepapers/eclipse-overview.pdf **[Accessed 29 October, 2005]**
9. OpenOffice Development Team. (2005). *System Requirements for OpenOffice.org* [online]. OpenOffice.org. Available from: http://www.openoffice.org/dev_docs/source/sys_reqs.html **[Accessed 01/11/05]**
10. Paul Browning. (2005). *TTW ("Through the Web") WYSIWYG Web Editors - The List* [online]. Genii Software. Available from: http://www.geniisoft.com/showcase.nsf/WebEditors **[Accessed 01/11/05]**
11. Paul Abraham. (2003). *WYSIWYG Editor for ASP.net* [online]. Abraham-Consulting. Available from: http://www.paul-abraham.com/ArticleDP/HtmlTextBox.htm **[Accessed 01/11/05]**
12. Writely *Writely - The Web Word Processor FAQ's* [online]. Available from: http://www.writely.com/BasePage.aspx?action=faq#faq0 **[Accessed 20/11/2005]**
13. Bill Gates. (1994). *Information at your fingertips.* Fall COMDEX.14 Nov,1994. Available from: http://www.microsoft.com/billgates/speeches/industry&tech/iayf2005.asp
14. Bjarne Stroustrup. (2006). *Bjarne Stroustrup's FAQ*. Available: http://public.research.att.com/~bs/bs_faq.html. **[Accessed 10/04/2006]**.
15. Walter Zorn(2005) *DHTML: Draw Line, Ellipse, Oval, Circle, Polyline, Polygon, Triangle with JavaScript* Available: http://www.walterzorn.com/jsgraphics/jsgraphics_e.htm [Accessed 22/04/2006]

Bibliography

Easton, M.& King, J *Cross-Platform .Net Development Using Mono, Portable .Net, and Microsoft .Net*

Gonzalez-Barahona,Jesus M. (). *Advantages of open source software* . Available: http://eu.conecta.it/paper/Advantages_open_source_soft.html. Last accessed 18/04/2006.

Robertson, James. (2003). *What is content management system*. Available: http://www.steptwo.com.au/papers/kmc_what/index.html. Last accessed 18/04/2006.
Writely . (2005). *Writely online tour*. Available: http://www2.writely.com/info/WritelyOverflowTourPage1.htm. Last accessed 10/04/2006.

Appendix

1. Entity relational diagram

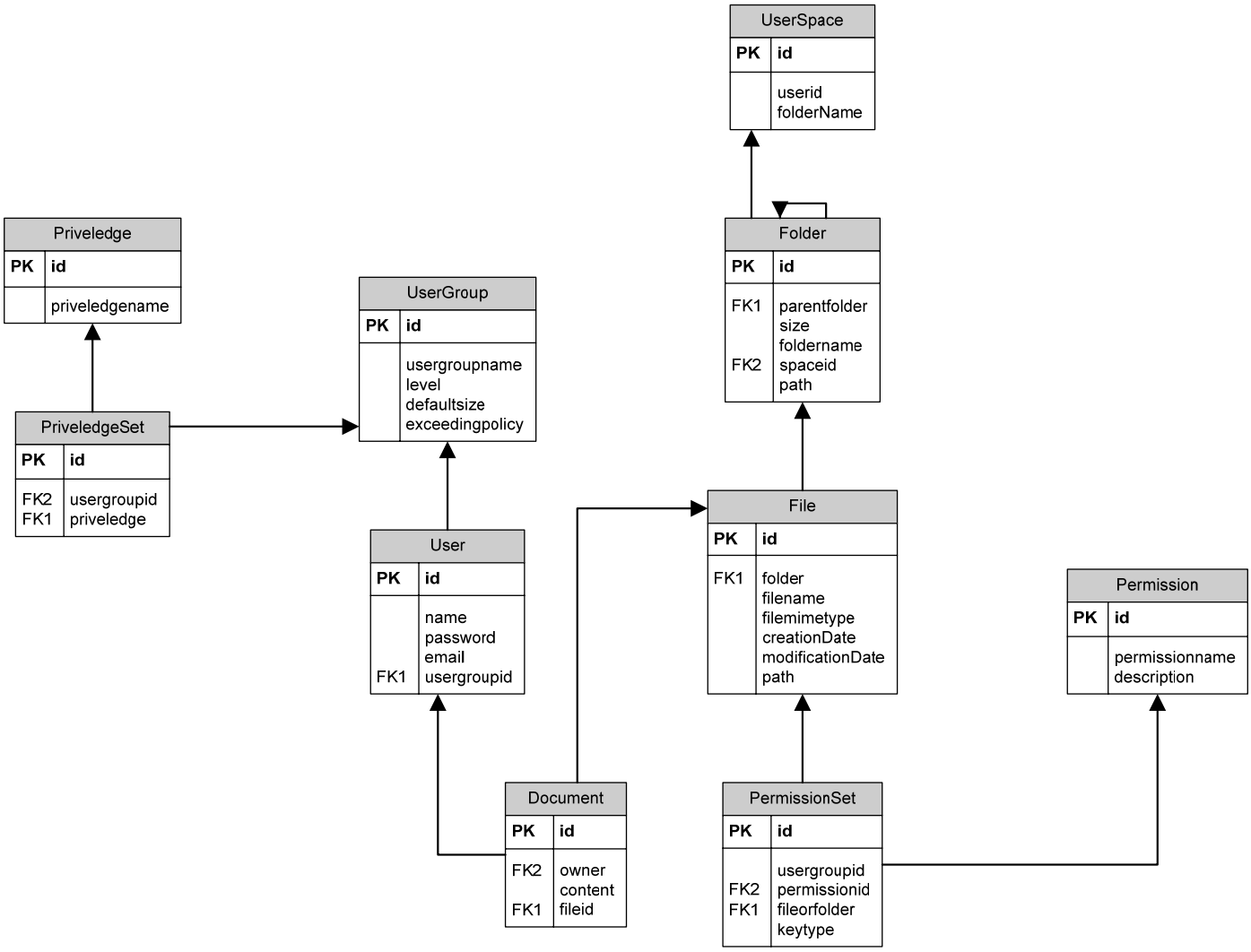


Figure 27 Entity Relational Diagram

2. Class Diagram

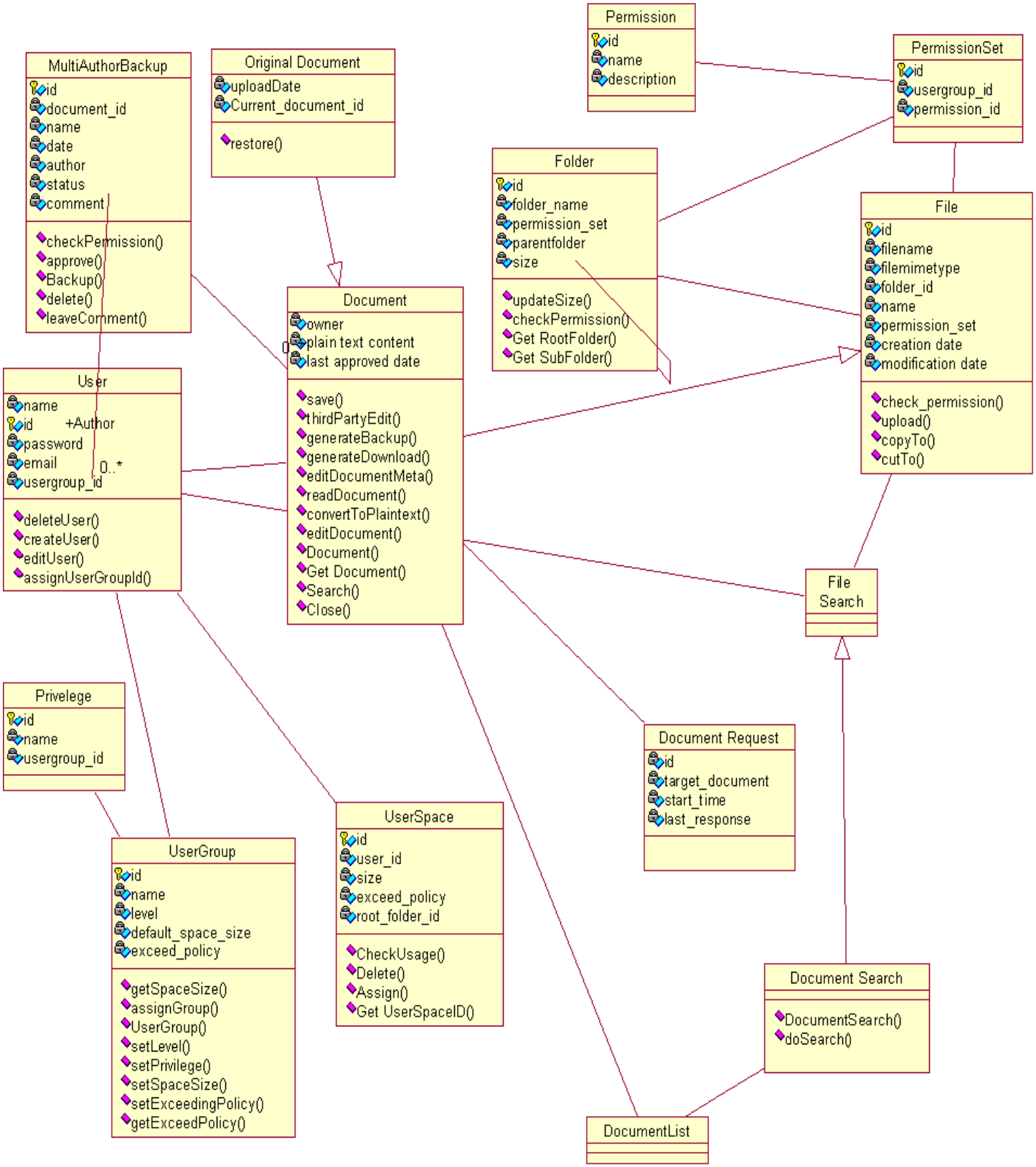


Figure 28 Class Diagram

3. Use Case Diagram

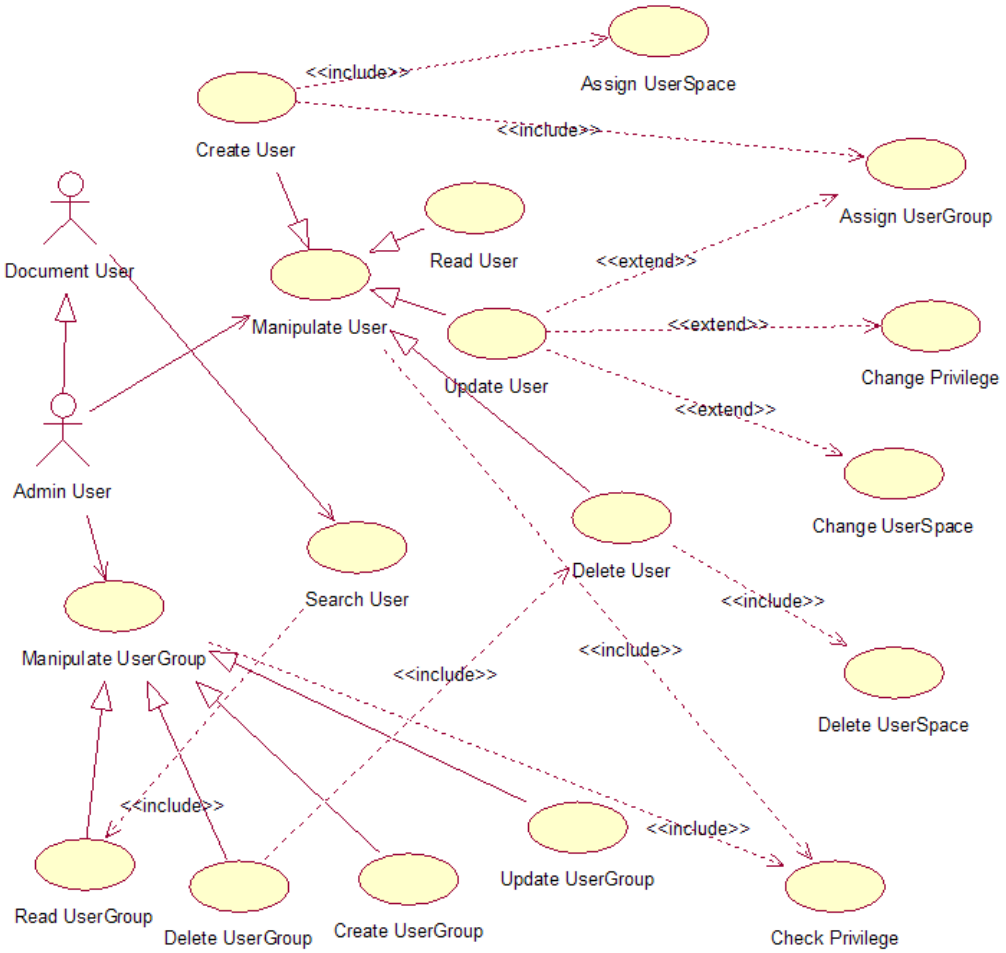


Figure 29 User System Diagram

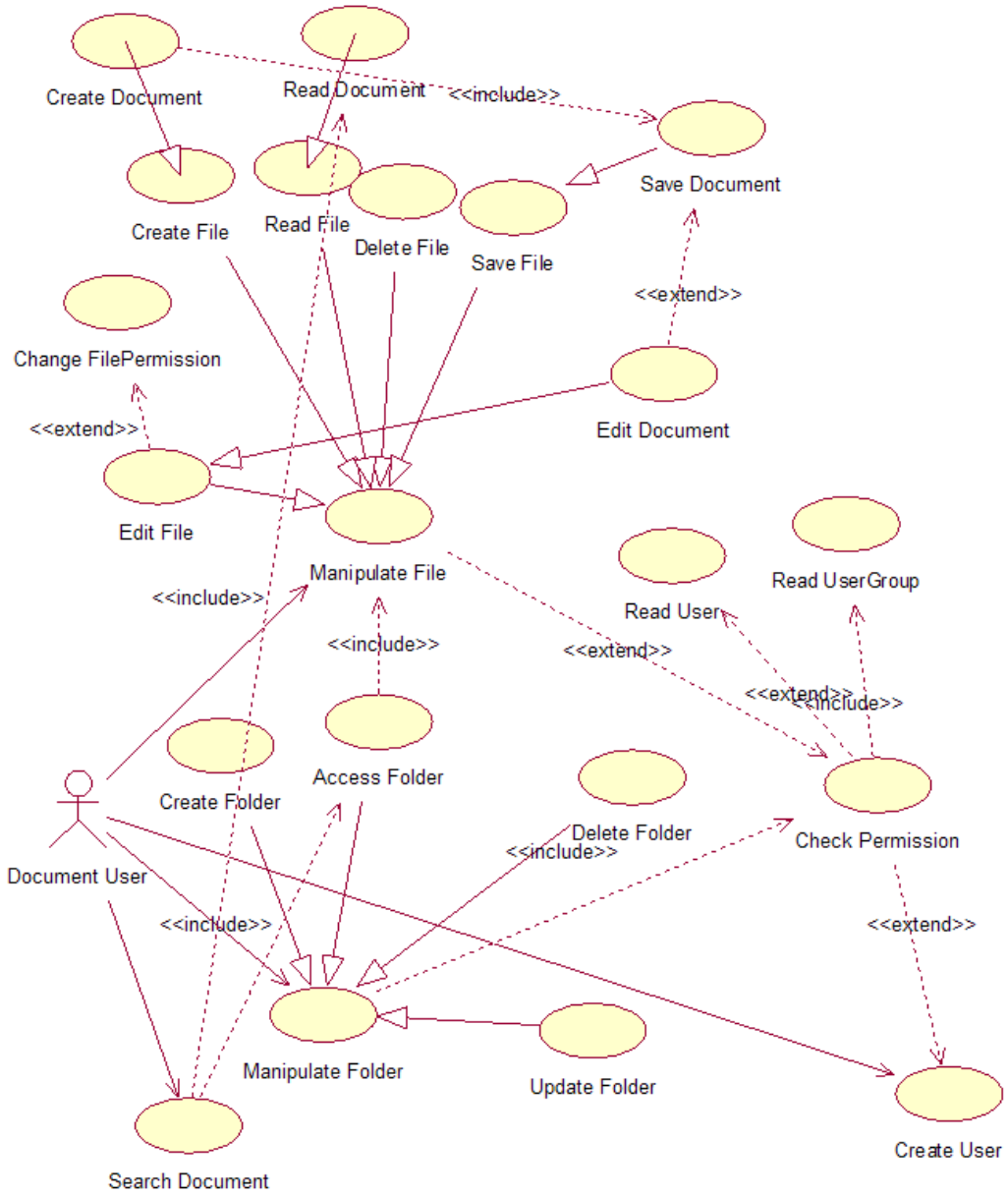


Figure 30 File System Diagram

4. HTML code for window

```
<div id="*w*frame"  style="BORDER-RIGHT:#eeeeee 2px outset; BORDER-TOP:#eeeeee 2px outset;
BACKGROUND:#cccccc; LEFT:130px; VISIBILITY:hidden; BORDER-LEFT:#eeeeee 2px outset; WIDTH:200px;
BORDER-BOTTOM:#eeeeee 2px outset; POSITION:absolute; TOP:150px; HEIGHT:240px; font-size: 12px;
padding-top: 14px;"></div>

<div   id="*w*titlebar"      style="BORDER-RIGHT:medium   none;   BORDER-TOP:medium   none;
BACKGROUND:#4455aa;    VISIBILITY:hidden;    OVERFLOW:hidden;    BORDER-LEFT:medium    none;
BORDER-BOTTOM:medium none; POSITION:absolute">

<span                                                                    id="*w*caption"
style="PADDING-RIGHT:0px;PADDING-LEFT:0px;FONT-WEIGHT:bold;FONT-SIZE:11px;LEFT:2px;PADDING-B
OTTOM:0px;COLOR:white;PADDING-TOP:0px;FONT-FAMILY:Verdana,Geneva,sans-serif;POSITION:relativ
e;TOP:2px; vertical-align: middle;"> *Title*</span>
</div>

<div id="*w*clientarea" style="BORDER-RIGHT:#cccccc 2px inset; BORDER-TOP:#cccccc 2px inset;
BACKGROUND:white;   VISIBILITY:hidden;   OVERFLOW:auto;   BORDER-LEFT:#cccccc   2px   inset;
```

```
BORDER-BOTTOM:#cccccc 2px inset; POSITION:absolute"></div>       <div   id="*w*resizehandle"
style=" position: absolute;"><img   src="Image/Window/winresize.gif" width="20" height="20"
alt=""> </div>
<div id="*w*resizebutton" style=" position: absolute;">
<img  src="Image/Window/button_up_outset.gif" width="16" height="14" alt="">
</div>

<div   id="*w*closebutton"   style="   position:   absolute;"><img     width="16"   height="14"
SRC="Image/Window/close_window.gif"></div>
```

5. JavaScript code assembling components

```
<script type="text/javascript">
<!--
//SET_DHTML("*w*titlebar"+CURSOR_MOVE, "*w*frame"+NO_DRAG, "*w*clientarea"+NO_DRAG,
"*w*resizehandle"+MAXOFFLEFT+210+MAXOFFTOP+90+CURSOR_NW_RESIZE,
"*w*resizebutton"+VERTICAL+HORIZONTAL,"*w*closebutton"+VERTICAL+HORIZONTAL);
//ADD_DHTML("*w*titlebar"+CURSOR_MOVE+TRANSPARENT);
ADD_DHTML("*w*titlebar"+CURSOR_MOVE);
ADD_DHTML("*w*frame"+NO_DRAG);
ADD_DHTML("*w*clientarea"+NO_DRAG);
ADD_DHTML("*w*resizehandle"+MAXOFFLEFT+210+MAXOFFTOP+90+CURSOR_NW_RESIZE);
ADD_DHTML("*w*resizebutton"+VERTICAL+HORIZONTAL);
ADD_DHTML("*w*closebutton"+VERTICAL+HORIZONTAL);
// Some vars to customize window:
var *w*frame_padding = 0;
var *w*titlebar_h = 19;
var *w*toolbar_h = 20;
var *w*statusbar_h = 20;
var *w*clientarea_margin = 4;
var *w*shown=1;
// preload button images to ensure un-delayed image swapping
var *w*button_down_outset = new Image();
var *w*button_down_inset = new Image();
var *w*button_up_outset = new Image();
var *w*button_up_inset = new Image();
*w*button_down_outset.src = 'Image/Window/button_down_outset.gif';
*w*button_down_inset.src = 'Image/Window/button_down_inset.gif';
*w*button_up_outset.src = 'Image/Window/button_up_outset.gif';
*w*button_up_inset.src = 'Image/Window/button_up_inset.gif';

// to save window height when window is minimized
var *w*last_window_h;

// *w*initWindow() moves elements to their adequate locations
// and builds coherences between these elements by converting outer *w*frame, client area and
images for resize functionalities
// to 'childern' of the draggable *w*titlebar
function *w*setTitle(title){
    document.getElementById("*w*caption").innerHTML=title;
}
function *w*initWindow()
{
    dd.elements.*w*titlebar.moveTo(dd.elements.*w*frame.x+2+*w*frame_padding,
dd.elements.*w*frame.y+2+*w*frame_padding);
    dd.elements.*w*titlebar.addChild("*w*frame");

    dd.elements.*w*titlebar.setZ(dd.elements.*w*frame.z+1); // ensure that *w*titlebar is
floating above *w*frame
    dd.elements.*w*titlebar.resizeTo(dd.elements.*w*frame.w-4-(*w*frame_padding<<1),
*w*titlebar_h);

dd.elements.*w*clientarea.moveTo(dd.elements.*w*frame.x+2+*w*frame_padding+*w*clientarea_mar
gin, dd.elements.*w*titlebar.y+*w*titlebar_h+*w*toolbar_h+*w*clientarea_margin);
    dd.elements.*w*titlebar.addChild("*w*clientarea");


dd.elements.*w*resizehandle.moveTo(dd.elements.*w*frame.x+dd.elements.*w*frame.w-dd.elements
.*w*resizehandle.w-2,
```

```
dd.elements.*w*frame.y+dd.elements.*w*frame.h-dd.elements.*w*resizehandle.h+1);

dd.elements.*w*resizebutton.moveTo(dd.elements.*w*titlebar.x+dd.elements.*w*titlebar.w-dd.el
ements.*w*resizebutton.w-*w*frame_padding-(*w*titlebar_h>>1)+Math.round(dd.elements.*w*resiz
ebutton.w/2)-18,
dd.elements.*w*titlebar.y+Math.round(*w*titlebar_h/2)-Math.round(dd.elements.*w*resizebutton
.h/2));

dd.elements.*w*closebutton.moveTo(dd.elements.*w*titlebar.x+dd.elements.*w*titlebar.w-dd.ele
ments.*w*resizebutton.w-*w*frame_padding-(*w*titlebar_h>>1)+Math.round(dd.elements.*w*resize
button.w/2),
dd.elements.*w*titlebar.y+Math.round(*w*titlebar_h/2)-Math.round(dd.elements.*w*resizebutton
.h/2));
    dd.elements.*w*titlebar.addChild("*w*resizebutton");
    dd.elements.*w*titlebar.addChild("*w*closebutton");
    dd.elements.*w*titlebar.addChild("*w*resizehandle");
    dd.elements.*w*titlebar.setPickFunc(*w*my_PickFunc);
     dd.elements.*w*resizebutton.setPickFunc(*w*my_PickFunc);
     dd.elements.*w*closebutton.setPickFunc(*w*my_PickFunc);
     dd.elements.*w*resizehandle.setDragFunc(*w*my_DragFunc);
     dd.elements.*w*titlebar.setResizeFunc(*w*resizeWindow);
     dd.elements.*w*titlebar.setDropFunc(*w*my_DropFunc);
     dd.elements.*w*resizebutton.setDropFunc(*w*my_DropFunc);
     dd.elements.*w*titlebar.hide();

}
*w*initWindow();

function *w*my_PickFunc()
{
    if (dd.obj.name == "*w*resizebutton")
    {
        dd.obj.swapImage(dd.elements.*w*clientarea.visible? *w*button_up_inset.src :
*w*button_down_inset.src);
        if(*w*shown==1){
        document.getElementById("*w*displayFrame").style.visibility="hidden";
        *w*shown=0;}else if(*w*shown==0){
        document.getElementById("*w*displayFrame").style.visibility="visible";
        *w*shown=1;
        }
    }
     if(dd.obj.name=="*w*closebutton"){
          dd.elements.*w*titlebar.hide();
    }
}

function *w*resizeWindow(width,height){
          width=parseInt(width)-45;
          height=parseInt(height)+15;
          dd.elements.*w*frame.resizeTo(width+2, height+2);
        dd.elements.*w*titlebar.resizeTo(width, *w*titlebar_h);

dd.elements.*w*clientarea.resizeTo(dd.elements.*w*frame.w-4-(*w*frame_padding<<1)-(*w*client
area_margin<<1),
dd.elements.*w*frame.h-*w*titlebar_h-*w*toolbar_h-*w*statusbar_h-4-(*w*frame_padding<<1)-*w*
clientarea_margin);

dd.elements.*w*resizebutton.moveTo(dd.elements.*w*titlebar.x+dd.elements.*w*titlebar.w-dd.el
ements.*w*resizebutton.w-*w*frame_padding-(*w*titlebar_h>>1)+Math.round(dd.elements.*w*resiz
ebutton.w/2)-18, dd.elements.*w*resizebutton.y);

dd.elements.*w*closebutton.moveTo(dd.elements.*w*titlebar.x+dd.elements.*w*titlebar.w-dd.ele
ments.*w*resizebutton.w-*w*frame_padding-(*w*titlebar_h>>1)+Math.round(dd.elements.*w*resize
button.w/2), dd.elements.*w*resizebutton.y);

dd.elements.*w*resizehandle.moveTo(dd.elements.*w*frame.x+dd.elements.*w*frame.w-dd.elements
.*w*resizehandle.w-2,
dd.elements.*w*frame.y+dd.elements.*w*frame.h-dd.elements.*w*resizehandle.h-2);
        document.getElementById("*w*clientarea").style.height=dd.elements.*w*clientarea.h;
```

```
}
function *w*my_DragFunc()
{
    if (dd.obj.name == "*w*resizehandle")
    {
        dd.elements.*w*frame.resizeTo(dd.obj.x-dd.elements.*w*frame.x+dd.obj.w+2,
dd.obj.y-dd.elements.*w*frame.y+dd.obj.h+2);

dd.elements.*w*titlebar.resizeTo(dd.obj.x-dd.elements.*w*titlebar.x+dd.obj.w-*w*frame_paddin
g, *w*titlebar_h);

dd.elements.*w*clientarea.resizeTo(dd.elements.*w*frame.w-4-(*w*frame_padding<<1)-(*w*client
area_margin<<1),
dd.elements.*w*frame.h-*w*titlebar_h-*w*toolbar_h-*w*statusbar_h-4-(*w*frame_padding<<1)-*w*
clientarea_margin);

dd.elements.*w*resizebutton.moveTo(dd.elements.*w*titlebar.x+dd.elements.*w*titlebar.w-dd.el
ements.*w*resizebutton.w-*w*frame_padding-(*w*titlebar_h>>1)+Math.round(dd.elements.*w*resiz
ebutton.w/2)-18, dd.elements.*w*resizebutton.y);

dd.elements.*w*closebutton.moveTo(dd.elements.*w*titlebar.x+dd.elements.*w*titlebar.w-dd.ele
ments.*w*resizebutton.w-*w*frame_padding-(*w*titlebar_h>>1)+Math.round(dd.elements.*w*resize
button.w/2), dd.elements.*w*resizebutton.y);
    }
}

function *w*my_DropFunc()
{
    if (dd.obj.name == "*w*resizebutton")
    {
        if (dd.elements.*w*clientarea.visible)
        {
            dd.obj.swapImage(*w*button_down_outset.src);
            dd.elements.*w*clientarea.hide();
            dd.elements.*w*resizehandle.hide();
            *w*last_window_h = dd.elements.*w*frame.h;
            dd.elements.*w*frame.resizeTo(dd.elements.*w*frame.w,
*w*titlebar_h+(*w*frame_padding<<1)+4);
        }
        else
        {
            dd.obj.swapImage(*w*button_up_outset.src);
            dd.elements.*w*clientarea.show();
            dd.elements.*w*resizehandle.show();
            dd.elements.*w*frame.resizeTo(dd.elements.*w*frame.w, *w*last_window_h);
        }
    }
}
//-->
</script>
```