Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Co-operative Authoring and Collaboration over the World Wide Web

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Technology

in

Computer Systems Engineering

at Massey University, Palmerston North, New Zealand

Daniel Robert Hean

2000

Copyright Daniel R. Hean © 1998-2000, all rights reserved.

Abstract

Co-operative authoring and collaboration over the World Wide Web is looking at a future development of the Web. One of the reasons that Berners-Lee created the Web in 1989 was for collaboration and collaborative design. As the Web has limited collaboration at present this thesis looks specifically at co-operative authoring (the actual creation and editing of web pages) and generally at the collaboration surrounding this authoring. The goal of this thesis is to create an engine that is capable of supporting co-operative authoring and collaboration over the Web. In addition it would be a major advantage if the engine were flexible enough to allow the future development of other access methods, especially those that are web related, such as WebDAV, WAP, etc.

Acknowledgements

Supervisor: W.H. Page, The Institute of Information Sciences and Technology

Supervisor 2: P. Lyons, The Institute of Information Sciences and Technology

Employer: School of Psychology, Massey University

Table of Contents

1 Introduction 1

2 Review of the literature and the (World-Wide) Web 5

2.1 The Web's first stirrings 5
2.1.1 The Vision and Original intentions for the Web 9

2.2 Web interaction 11
2.2.1 Typical current web sites 12

2.3 Summary 15

3 Collaboration and the Web 17

3.1 Co-operative authoring and Collaboration 17

3.2 Co-operative authoring and the Web 18
3.2.1 The future of the Web 19

3.3 Collaboration today 24
3.3.1 Collaborative editing and version tracking packages 24
3.3.2 Version tracking information 25
3.3.3 Currently available options 25
3.3.4 Web collaboration today 29
3.3.5 Lost Versions 30

3.4 Types of Web publishing 32

3.5 The future of the Web 35
3.5.1 Increasing the success of the Web 37

3.6 One possible answer 37

4 Design through to testing 41

4.1 Design 41
4.1.1 Web-based access 46

4.2 Design of the engine 48
4.2.1 Java Servlets 54

4.3 Current progress 55

4.4 Testing 56
4.4.1 Creating a testbed 57

4.5 Summary 59

5 Conclusion 61

6 Glossary 67

7 Appendix 75

7.1 Appendix A: A full example of a preference file 75

7.2 Appendix B: Using YEdit 77

7.3 Appendix C: The Middle Interface 81

7.4 Appendix D: The Server Interface (File System example) 89

8 References 91

Table of Figures

Figure 1: Relationship models 15
Figure 2: Example web page for Web themes 43
Figure 3: Example of the Web theme data 43
Figure 4: Interaction of the main components of YEdit 48
Figure 5: The Web Interface servlets 49
Figure 6: The toolbar that appears at the top of the document in Browse mode 50
Figure 7: The edit screen 51
Figure 8: Example portion of an initialisation file 58

1 Introduction

When I first discovered the Internet, back in the days before there was any overt commercial interest in the Internet, it was primarily a link between universities and other non-commercial groups. At that time one of the main uses of the Internet was for collaboration between people on a variety of different subjects. This collaboration was primarily based on text communications such as email, Usenet, and talk. They were mainly text based because of the limits of the bandwidth available at the time, although there were probably some groups connected with, what at that time would have been called, high speed. An example could be a high-speed backbone of 56Kbps, which was very high speed, when compared to some of the modem connect speeds at the time (for example 300 baud, that is 300 bits per second).

When I started to investigate the origins of the World Wide Web [Berners-Lee WtW], I came across some similar ideas for its use, that is, for collaboration on other projects. There was more to it than just collaboration on other projects though. Included in Berners-Lee's original visions for the Web was the idea of collaborating on and about the Web itself. This combined with other ideas from other areas (such as the open manner of collaboration that I had experienced over the Internet, older methods of group document creation and editing, etc) led me to look at the co-operative authoring of documents. This research concentrates on the co-operative authoring of documents, which in terms of this research means the actual collaborative work on a document. This can be compared with collaboration, which is one of the support mechanisms that allow people to work on documents together which will be looked at, but is not the main focus. This is an important distinction for this document, as it is specifically focused on the co-operative authoring of documents, rather than all the support mechanisms that surround that authoring (such as email, Usenet, discussion groups and the like).

The first area that is examined is co-operative authoring and collaboration in relation to the past and future of the Web. The second (and main) purpose of this research is to design and implement a vision of co-operative authoring and collaboration over the Web. Part of this vision dates back to the original proposal for the Web [Berners-Lee 1989].

The vision that I have for the future of the Web allows everyone the freedom to publish to the Web at anytime, from any location, using their favourite tool. Also to do so in a manner that allows co-operative authoring of documents and collaboration between the people involved, including full co-operative authoring of documents, rather than just sharing them. This is the basis for what I am calling a free flowing web of information. This may sound like an oxymoron, because a web is typically static, while it catches things that flow through it. In the sense of the Web though, information is flowing around it (currently not freely, as it mostly only flows from a webmaster to a web surfer), but at the same time, the Web is not static. The Web is continually changing, which is one of the challenges that people can face when trying to understand it. For example search engines and web maps will probably never cover the whole of the Web, because it is continually expanding at a tremendous rate, while other parts disappear. This means that not only is information moving through the Web, the Web itself is moving and changing as parts are changed, added and removed.

In conventional approaches to creating and publishing documents on the Web, one person or group assembles the information and then publishes it in "read-only" format on the web site. Many authors modify this idea and use automatically generated content, such as constantly updated prices, or a threaded web discussion. Generally this is separated from the rest of the site, either by position, or by some marking out. Some of this automatically generated content may be an integral part of the site, although this tends to change less often, for example, menus, page headers and page footers. The fully automatic generation of content and/or pages can be quite clearly defined as different from the static web pages (even though the distinction does blur a bit when you talk about server executed pages, such as SSI (Server Side Includes) or Java Servlets (Java [Java] running on the server), which include pieces of automatically generated content in otherwise static pages). The reason that fully static pages and fully dynamic pages can usually be quite clearly defined is primarily because the type of information present is quite different. Static pages typically have information that does not change all that often, whereas dynamic pages have information that can be changing continuously, or at least very often. This distinction can blur with the use of static pages as cache for a dynamic process (as can be the case in multi-tier web sites), or with the use of server executed pages where part of the page is static, and part is dynamic, which means that the page can contain both static and dynamic information.

The antithesis of the normal web publishing method is to allow anyone and everyone to change and update the web site. An example of this is WikiWiki [Cunningham], a web site created and maintained by W. Cunningham at http://c2.com/cgi-bin/wiki that implements some of the co-operative authoring vision for the Web and allows for a limited amount of free flowing of information. It can be described in many ways, such as this quote from its home page (current October 2000) "(It is) ... a fun way of communicating asynchronously across the network", or as a set of web pages that are open and free for anyone to edit as they wish. Wiki is not real-time; therefore people have time to think before they follow up a web page, often days or weeks, so they have time to consider what they write. It was created for the discussion of People, Projects, and Patterns (a pattern is a recurring solution to a common problem in a given context and system of forces [Alexander 1977][Alexander 1979]).

The mixing of ideas and possibilities that can occur in an environment such as WikiWiki is in essence very similar to how the Internet came into being, and it is a good example of one way of implementing a web site that is based on the free flow of information. It is completely free flowing, and follows no set rules; anyone can update anything, anytime, anywhere, with no barriers stopping people from doing what they want. This free flowing method of web publishing could be anarchy when compared with the current normal web publishing practice, but it shows up some interesting comparisons with the normal method of publishing on the Web. On a normal web site the expectation is that only those changes that are approved and have gone through the normal channels will ever be made to the web site. This can have the consequence that when a change that is detrimental for the web site is made through approved channels, it can take just as long to fix it (once it is known), as it took to implement. Another consequence is that because any changes have a set channel to go through, the owner can be lulled into a false sense of security that no unauthorised changes will ever be made. [Risks] Thus, if unapproved changes are made, it can cost a tremendous amount of time and money to fix. The free flow of information on a web site can also be the key to realising a whole new form of web collaboration. It is based partly on the

original vision for the Web (that has not yet been fully explored) and extends it using today's ideas and tools while keeping the compatibility to allow the majority of people on the Web to have access to and use it.

I have a vision of the Web as a free flowing web of information that flows in any direction. The implementation of this vision provides the freedom for people to have their views heard and seen by those that need to see them most. It enables people with unique ideas to co-author documents in an easy and commonly available format, and it allows for anyone to use the tools they are most productive with, at the time of day or week that they are the most productive.

The author has been working on a web site (http://www.YEdit.com/) that gives an overview of the ideas for a free flowing web, as well as an implementation. Web sites can be set up so that multiple people can update the open part of the sites, while groups can share information, and co-operatively and collaboratively write documents. This web site fits in nicely with the current crop of CSCW (Computer-Supported Co-operative Work), BSCW (Basic Support for Co-operative Work) [BSCW] and Groupware sites that are set up to enable collaboration about documents.

2 Review of the literature and the (World-Wide) Web

This research is concerned with co-operative authoring of documents and collaboration about those documents, specifically over the World Wide Web. Secondary to this is whether the research can be generalised to support co-operative authoring using systems of communication other than the Web, to allow for future expansion. This purpose is supported by Berners-Lee's original intentions for the Web, which includes support for collaborative authoring and collaborative design of something other than hypertext itself. Because of this we shall therefore review the original vision that drove the development of the Web, and the Web, as it is today.

Berners-Lee initiated the creation of the Web in 1989, when he wrote and circulated for comment a project proposal [Berners-Lee 1989] for a networked Hypertext system for CERN (Conseil Européen pour la Recherche Nucléaire (French), European Organisation for Nuclear Research, now known as the European Laboratory for Particle Physics). The project proposal was based on the need to enable people to share documents, work, and ideas across groups, and allow interaction between those groups.

Although the Web has been spectacularly successful in its current form, has it really lived up to its original goals? Is the current form of the Web all that it was intended to be, or was it intended to be something more? To answer these questions, we need to go back in time to both some initial stirrings about the possible use of technology, and of course the conception of the Web by Berners-Lee in 1989, a mere eleven years ago.

2.1 The Web's first stirrings

'*As we may think*' [Bush 1945] details Dr. Bush's view of the future of science after the war. It details the things that scientists should work on to bring in a new age. It goes into detail about technology that we use in our everyday lives, in terms of the day. The technology that Dr. Bush mentions in this article is very forward thinking and includes things that resemble our current computers, digital cameras, mass storage of information (such as mini-hard drives, CD's, and DVD's) on small sized media, speech recognition systems, credit cards, the Web and much more.

The **Timeline** below was condensed primarily from the following two sources:

- "A Short History of the Web" [Cailliau 1995]
- "A Little History of the World Wide Web" [W3 1995]

Pre Web creation history (1945-1989)

1945 July	"As we may think" [Bush 1945], by Vannevar Bush
...	
1965	The term "Hypertext" is coined by Ted Nelson
1968	The first Hypertext system is produced and demonstrated by Douglas Engelbart on the 9th December at the Fall Joint Computer Conference. Along with this, other innovations such as the mouse and shared computer screens were demonstrated.
...	
1979	SGML (Standard Generalised Markup Language) is invented by Charles Goldfarm. HTML (HyperText Markup Language) is based on SGML.
1980	Berners-Lee creates "Enquire" [Berners-Lee WtW] a notebook program, "Enquire-Within-Upon-Everything", which allows links to be made between arbitrary nodes. Each node had a title, a type, and a list of bi-directional typed links.
1981	In "Literary Machines" Nelson [1981] describes a worldwide publication system ("Xanadu").
1987	CERN and US laboratories connect to the Internet as the main means of exchanging data

This timeline gives some indication of the events that led up to and influenced Berners-Lee's proposal. As can be seen, there are several events that preceded Berners-Lee's proposal to create what was later named the World Wide Web (until 1990 it was named 'Mesh'). The pre-web time scale, starting from Bush's article, is considerably larger (1945-1987, 42 years) than the post-web time scale below (1989-1995, 6 years). This is because after the Web was first introduced, it picked up momentum very quickly, and has kept increasing its momentum to this day. The timeline below details some of the important steps that have led the Web to where it is at the present time.

Post Web creation history (1989-1995)

1989-March	First project proposal for a networked Hypertext system for CERN (the Web is born) written and circulated for comment (Berners-Lee)
	Paper "HyperText and CERN" produced as background
1990-September	Mike Sendall, Berners-Lee's boss, OKs the purchase of a NeXT cube processor, and allows him to go ahead. He selects "World Wide Web" as a name for the project (over Information Mesh, and others)
1990-November	Initial WorldWideWeb prototype developed on the NeXT (Berners-Lee)
	Nicola Pellow joins and starts work on a line-mode web browser
	Berners-Lee gives a colloquium on hypertext in general
1990-Christmas	Line mode and NeXTStep web browsers demonstrable
	Access is possible to hypertext files, CERNVM "FIND", and Usenet
1991-May-17	Presentation to C5 committee and general release of WWW on central CERN machines
1991-June-12	CERN Computer Seminar on WWW
1991-August	Files available on Usenet, posted on alt.hypertext (6, 16, 19th Aug), comp.sys.next (20th), comp.text.sgml and comp.mail.multi-media (22nd)
1991-October	VMS/HELP and WAIS gateways installed
	Mailing lists www-interest (now www-announce) and www-talk@info.cern.ch are started
	One year status report
	Anonymous telnet service is started

1992	The world has 50 web servers
1992-January-15	Line mode web browser v1.1 is available by anonymous FTP
	Presentation to AIHEP'92 at La Londe
1992-February-12	Line mode v1.2 is announced on alt.hypertext, comp.infosystems, comp.mail.multi-media, cern.sting, comp.archives.admin, and mailing lists
1992-July	Distribution of WWW through CernLib, including Viola. WWW library code ported to DECnet
	Report to the Advisory Board on Computing
1993	50 Web servers grows to 250
	The Mosaic browser is created
1994	2500 Web servers
1995	73500 Web servers
	Sun creates HotJava

The following sums up the Web, as it was just a couple of years after it was created.

WorldWideWeb – Summary (1991-1992)

The WWW project merges the techniques of networked information and hypertext to make an easy but powerful global information system.

The project represents any information accessible over the network as part of a seamless hypertext information space.

W3 was originally developed to allow information sharing within internationally dispersed teams, and the dissemination of information by support groups. Originally aimed at the High Energy Physics community, it has spread to other areas and attracted much interest in user support, resource discovery and collaborative work areas. It is currently the most advanced information system deployed on the Internet, and embraces within its data model most information in previous networked information systems.

In fact, the Web is an architecture which will also embrace any future advances in technology, including new networks, protocols, object types and data formats.

http://www.w3.org/Summary.html (Berners-Lee)
W3 Status: Historical interest

2.1.1 The Vision and Original intentions for the Web

Berners-Lee attempted to persuade CERN management that the Web was in their best interests. He detailed in this document entitled "Information Management: A Proposal" [Berners-Lee 1989] the original proposal that was the first step in the creation of the World Wide Web. It details some of the problems that CERN was having (especially information management, and tracking of large projects), and makes a proposal to create a system that was the beginnings of the Web.

The following quote shows what the Web was originally designed to be used for. It details the list of intended uses for the Web at CERN. Most of these, such as online encyclopaedias, online help and documentation, major news organisations, personal homepages, and to a lesser extent collaborative work [Berners-Lee 1990] are currently being used.

Intended uses for hypertext at CERN

General reference data (encyclopaedia, etc)

Completely centralised publishing (online help, documentation, tutorial, etc)

More or less centralised dissemination of news which has limited life

Collaborative authoring

Collaborative design of something other than the hypertext itself

Personal notebook

http://www.w3.org/DesignIssues/Uses.html (Berners-Lee)
W3 Status: W3 Archive

From a talk in 1991 and 1992 entitled "W3 Concepts" [Berners-Lee 1991], the following concepts for the Web (that follow on from the intended uses that CERN had for hypertext), were put forward at an online seminar about the World Wide Web Consortium (http://www.w3.org/).

In the talk Berners-Lee talks about universal readership that allows anyone to retrieve information from any computer that they are using at the time; rather than having to go to a specific computer in a specific location to retrieve the information. Hypertext essentially allows documents to have links in them that point to other locations (typically other documents). This allows the restriction of reading documents in a serial fashion to be removed. With the removal of this restriction, documents can have links to and from any place, allowing direct linking to appropriate information, such as indexes, bibliographies, and other media such as graphics, sound and video. Searching allows for the location of specific information by the use of an engine behind the pages; this searches for the requested information and returns the results as a web page with links to the pages that it found. The client/server model allows for distributed clients and servers that are linked together by a protocol, and creates a decentralised system so that anyone can set up a client to read information, and anyone can set up a system to

serve information. Format negotiation allows the client and the server to set up formats that both understand. This gives the server the ability to serve the most appropriate information to the client, such as supplying the client with graphics that it understands, or giving the information in a language that both understand.

By the stage Berners-Lee delivered this talk, the Web had started to take off. However as is shown by the quote below, some of the main reasons for creating it had been left behind. The realisation of a good HTML editor took longer than was expected (and the quality of the output still varied widely).

The Future (1993)

> The WWW initiative has taken off and become the emerging leader in Internet information systems. However, it has been overtaken by its popularity, and many of the original design goals for a collaborative tool have still not been implemented.
>
> At the same time, it is spreads(sic) into many fields which put demands on its functionality. Fortunately, these all fit in well with the original design concepts.
>
> Collaborative work was the original design goal of W3. This involves everyone working together in a group to be able to share knowledge, modifying, annotating, and contributing as well as reading. This is an exciting area. It requires good wysiwyg hypertext and hypermedia editors (which will probably arrive during the next year, 1994) as well as authentication of users.
>
> http://www.w3.org/Talks/CompSem93/FutureText.html (Berners-Lee)

One of the original visions behind the Web was to provide a collaborative computing environment that provided an easy method for the exchange and storage of information and knowledge. Information and knowledge were not meant to become lost in a sea of web servers and surfers, with people struggling to find the information that they require. As the Web has grown, and outstripped other methods of accessing information over the Internet [NetValue 2000], many people form an idea that the Web is all that there is to the Internet, as that is the part that they see most often.

Looking at statistics for Internet usage as at November 2000 shows clearly that Web use is higher than other uses of the Internet. The percentage of people with Internet access that browsed the Web in past month in the USA was 97.3% and in the UK was 96%, whereas the percent of people that used email in the USA was 44.5% and in the UK was 60.6%. Usenet usage was even less at 5.1% in the USA and 10.5% in the UK.

Some of the ideas for the Web (http://www.w3.org/DesignIssues/Editor.html April 1998) were that it should be universal and be able to encompass anything, from a scribble on the back of an envelope, through to a polished work of art. It should easy to work on documents, add links, annotations, and to copy information containing embedded back links to the original document.

The Web has several key features that have helped to propel both its initial acceptance, and its growth since that time.

The simplicity of the Web's client/server design allows for ease of distributed use and development, allowing people to use it easily, regardless of location. The Web can leverage legacy technology, which means that people can still access and use legacy technology with the latest web browsers, for example Gopher [RFC1436] and WAIS [RFC1625]. The Web is platform- and operating system- independent, which means that people are free to choose the system that they want, rather than being locked into using only one system. Because the Web is based on open standards, we know that it will still be available in the future, whereas if it were proprietary, users would be locked into both the whim and success of the proprietor company. The Web standards are also portable, extensible and scaleable, which means that the standards can be adapted and changed to changing circumstance, and they will tend to have greater acceptance because as they are open and portable, there are more people that can use them.

2.2 Web interaction

Commonly the Web is used for retrieving information, and there is very little in the way of interaction between users. Information transfer over the Web is mostly one way, from the Webmaster, to the reader. There are a few techniques to make the Web more interactive.

One method is to add a section to the web site that gives people a form that they can fill in with appropriate information, and then submit it. These can be used for email feedback, for entry into guest books that appear on the page, as general comment forms that get saved into a database, or as more interactive things such as polls and visible comments on documents.

Another method of making the Web more interactive is by setting up customisation features for users. This can be done by storing information about how the user wants information presented to them, what information they want presented to them, and also whether they want to be alerted to new or changing information, for example being alerted by email when a web page changes.

A different method that is employed by a small number of web sites, such as WikiWiki [Cunningham], and its clones, is to make the whole web site editable by anyone. This opens up the whole site to being edited and changed by anyone (although it could be limited to a select group), and allows changes directly to the web pages.

In order to understand why co-operative authoring will be useful, one needs to have some understanding of the current state and sophistication of web sites. Company web sites are looked at because they have a good range of sophistication that ranges from web sites that show very little knowledge of the Internet and the Web, through to companies that use the Web to the fullest.

2.2.1 Typical current web sites

Looking at typical current web sites gives a good background to what is currently used by a reasonable proportion of web sites. This is good background because it allows one to compare the differences between the current use of the Web, and the abilities that are being proposed and implemented for the research for this thesis.

Current web sites are predominately created to serve content or to enable e-commerce for users over the Web. There are many examples of web sites with significant content that is very heavily used, such as good search engines that have a large amount of information. There are also many examples of web sites that have little in the way of content; for example many personal home pages, which just have the person's name and not much more. As there is little content, there is no compelling reason for anyone to come back (as can be seen by the low number of hits commonly displayed on their page counters).

Currently the main use of the Web is the transfer of information. This can be simply to make any information available, just in case someone might want that information (for example many personal home pages, or a simple 'Hello, this is who I am...'), or to make specific information available that the general public wishes to know (whether it is domain specific information, entertainment, or something else). The creators of web sites need to think about the information they are willing and able to give their visitors. After all, if a visitor doesn't get something from a web site (from information that they need, through to a warm fuzzy feeling, if that is what the web site has been created for), they will not come back. What's more, they will not refer other people to the web site. Many people believe "Word of mouth", is one of the most powerful forms of publicity.

Typical progression as companies discover(ed) the Web

Many companies start with little or no knowledge of either the Web or the Internet, and progress through stages as they learn more about the Internet and the Web; and realise the impact that it is possible to create with good use of these technologies.

- **Individual user sites:** At this stage there are just a few people in the company who are either aware of the Web, or have an idea of what can be achieved using the Web. These groups can put information onto the Web that they think would be useful and that they are familiar with. This means that there are a few sites that give a taste of the company, but they are commonly separate with few links between them, and often display duplicated information. This method of creating awareness of the company on the Web is a good start and has the potential (especially today) to quickly show positive returns, now that the Web has more awareness throughout the community.

- **Brochureware:** Brochureware sites are easy to distinguish from other sites because they are commonly a direct copy of a print brochure that has been scanned and placed on the Web with little or no changes made. This kind of site is very easy and cheap to create, but it is also deceptive in that it doesn't really achieve much, especially when it only contains phone numbers and no email addresses, as is common.

- **Limited interaction:** Limited interaction sites contain useful information about the company, products and the people that make up the company. This is probably the most common type of site. Often this is all that a company needs, especially if it is not their business to sell products. On sites such as these, there could be anywhere from just a page or two (for a small company) through to hundreds or more pages (for a larger company). There are generally different methods displayed for contacting the company, from their physical address, phone numbers, email addresses, feedback/comment forms, and sometimes other contact methods such as online chatting.

- **E-Commerce:** E-Commerce sites generally contain all the information that a limited interaction site contains, with the added ability to buy and sell directly from the web site. Sites that implement e-commerce have either a product or a service to sell and they are generally designed so that the process of buying and selling is the primary focus of the web site.

- **Interactive and relationship building:** Interactive and relationship-building sites generally focus on interactive features with the customer, and building an ongoing relationship with them. This can consist of many different ideas and processes, all of which are in place to keep the company's name and products in front of the customer as much as possible, with the customer's permission. These sites can have mailing lists, discussion pages, online chats, and many other options for the customer to interact with the company, and in the process build a higher awareness of the company in the customer's mind. These sites can also be built for companies that do not sell services or products (for example news sites) that want to ensure constant and repeat visitors, while providing them with a high quality service.

Unfortunately, not all companies that create a presence on the Web understand what the Web and the Internet are, and how they work. Typically, such companies and individuals are unaware of Netiquette, which is the social conventions that are necessary if the net is to remain a pleasant working, educational, and recreational environment. Consequently they are inclined to treat the Internet as their own exclusive resource, and abuse its ability to deliver information to millions of uninterested computer users. By abusing the power of the Internet they force others to pay for their bad marketing campaigns by sending hundreds of thousands of emails, or newsgroup postings extolling their products and services. This unfortunately is quite prolific at the moment, and it is also the most destructive. This is commonly called 'SPAM' (It got the name from the Monty Python script in which there is the repeated overuse of the product called 'Spam', a potted meat. In this script a customer comes into a shop to buy some food, and everything contains many portions of Spam, and little else. The script ends in a song that only contains the word 'Spam').

Some other common faults are:

- Web sites/pages that only have contact details to be contacted for more information. It is a better idea to supply the information on the web page along with contact details, so that people can access the information, and then follow up with contact if further contact is necessary.

- Pages that require you to fill in a form before they will post, email or otherwise contact you with information that can readily be displayed on their web page

- Pages that have a lot of fluff, and/or little or no content/information.

- Pages that are over laden with graphics or scripts (most users are connected by modem. In the USA 49.9% are connected but only 6% have a broadband connection, in the UK 31.2% are connected but only 1% have a broadband connection [NetValue 2000] and even those that do have a broadband connection, may want to see the page quickly, as they don't have the time to wait around).

- Pages that require the use of graphics/JavaScript/etc, or only display on the latest web browser (This can be annoying for the average user who is browsing, but can be discriminating for those with disabilities. For example if the web site is only accessible with graphics, a blind person can't use it. Web sites that able-bodied people have trouble accessing, can be even worse for those with disabilities).

If a company can avoid most of the common faults, it demonstrates that they have a good understanding of the Web and the people who use it. When companies have a good understanding of the Web and the people who visit their site it is much easier to create a site that will meet the needs of the visitors and entice them to come back. It is always a good sign to have most visitors as repeat visitors, as they already know the site and have an idea of whom you are which makes it easier to achieve the aim of the site. Unfortunately it is very easy to get led off in another direction by nice "eye-candy", rather than following the main aim for the site. This is bad for both web site developers, because it is unproductive work, and it is bad for the visitors, as it provides a distraction from the reason that they have come. Also another downside to "eye-candy" is that often, specific browsers and/or plug-ins are required to display it, which means that it can easily destroy the usability of the site for most other users. Generally if a company reaches this point they will have put together a small group that takes responsibility for the corporate web site, and will keep it updated and expanding. Generally when companies put a small group together to work on a web site, it is with the understanding that all changes to the web site go through that central group. This leads into the different types of models for web sites, as the One-Many model described above, is not the only model for web sites.

The common model for web sites in companies is a One-to-Many relationship (See Figure 1: Relationship models). That is, there is one person (or group) who is responsible for the web site, while there are many people viewing the site. This is in contrast to the original objective for the Web, with Many-to-Many relationships where many different people are discovering the Web and creating sites by themselves. This

One-to-Many relationship is a common thread throughout companies, and has been tried and tested over a long time, and it does work, when implemented correctly.

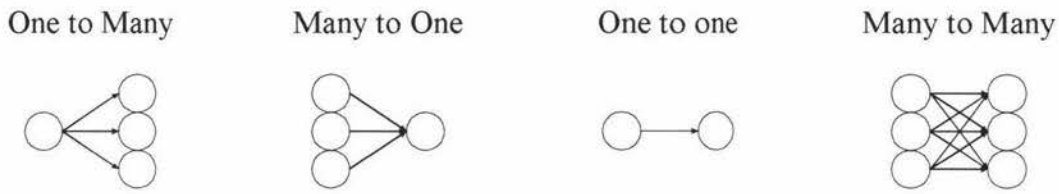


Figure 1: Relationship models

The other side of the coin, the Many-to-One aspect is commonly found as part of a larger overall site. A good example of this is where people can leave feedback for the web master, or other designated person. One other place to find Many-to-One sites is as part of a Many-to-Many site, where there are people working together on a project. As part of their site they have information that is appropriate for a manager, or other people further up the chain of command. This type of site allows collaboration between group members who will commonly use many other methods to collaborate, rather than just the Web. The web site is likely to be a small part of the collaboration process.

2.3 Summary

Many people now think that the Web is all that there is to the Internet, and many think that the Web *is* the Internet. Because the Web has taken off, and become the main activity on the Internet [NetValue 2000] (compared with FTP, email, etc), its popularity has overshadowed the fact that there is a lot more to the Internet than the Web application that runs over the Internet. There are another two reasons why the Web has seemed to overtake other applications on the Internet. The first is that almost all web browsers support connections to other Internet protocols, such as GOPHER, WAIS, and FTP, as well as supporting other common Internet standards, such as Email, and Usenet (Newsgroups). The second significant reason that the Web is seen to be the most popular part of the Internet is that it is very visible, and has had a lot of publicity to actively promote it. Other equally important parts of the Internet, running as server programs and applications, are invisible to most users. The hidden side of the Internet is seldom seen because it provides the infrastructure that runs everything, rather than the flashy applications that people use. It is this infrastructure that provides the common connections, protocols, and systems that the user applications rely on.

Berners-Lee created the Web in 1989 to enable people to share work and ideas in an easy-to-use form. It was based on ideas and developments dating back to at least 1945 with Vannevar Bush's article 'As we may think' published in the Atlantic Monthly in July 1945.

Since 1989 the Web has come a long way in terms of publishing information and one-way communication. Unfortunately, in this process only some ideas have been pushed forward, and some ideas like the idea of co-operative authoring and collaboration, which was one of the cornerstones of its creation, have not progressed very well.

Currently there are some options available for collaboration on documents over the Web, but most of them rely on all authors using the same editing program, or they are limited to a specific domain of use. There are some web based document collaboration sites, but these are limited to a specific domain, and require the use of specific notation and are good for the niche that they inhabit. Currently the options for full and open co-operative authoring (including creation of web sites) are very limited, especially with regard to the packages that can be used.

3 Collaboration and the Web

This research concentrates on giving people the ability to collaborate on documents to allow group creation and editing, specifically co-operative authoring of documents. One of the driving forces behind using the Web is to use a technology that is currently widely available and accessible by anyone, anywhere at anytime.

3.1 Co-operative authoring and Collaboration

To understand why this research has been called "Co-operative Authoring and Collaboration over the World Wide Web", one must look at the definitions of the words (in particular the group, "Co-operative Authoring and Collaboration"). Although at first sight the phrase "Co-operative Authoring" and the word "Collaboration" may seem to mean the same thing, they in fact mean different things, and they are also different in terms of the scope of the definition. The following definitions in *italics* are from the Merriam-Webster dictionary [Merriam Webster]. When the word is defined in terms of a similar word, that similar word has also been defined.

Co-operative: *a) marked by co-operation <co-operative efforts>. b) marked by a willingness and ability to work with others <co-operative neighbours>*

Co-operation: *the action of co-operating: common effort*

Co-operate: *to act or work with another or others: act together*

Authoring: *to be the author of*

Author: *a) one that originates or creates. b) the writer of a literary work (as a book)*

Collaboration: *to work jointly with others or together especially in an intellectual endeavour*

By stringing the definitions for the terms together, it is easy to see that the dictionary definition for "Co-operative Authoring" is "marked by a willingness and ability to work with others to be a writer (or creator) of a literary work". In terms of this research the literary work is a web page, but using the YEdit engine, it could be any kind of document.

Collaboration is easier, as it is not defined using similar terms. Collaboration means "to work jointly with others or together especially in an intellectual endeavour".

As can be seen by these definitions, Co-operative Authoring is group creation of a literary work (which in terms of this research means a single, or group of web pages), whereas Collaboration means to work with others, which covers a much wider group of

activities, which may (and will in this research) include Co-operative Authoring, but also covers a diverse set of other collaboration activities.

Both the terms co-operative authoring and collaboration are required to describe the content of this thesis, because although the main focus is on the co-operative authoring of individual documents, there is also the aspect of collaborating with others to create the whole site (or portion thereof).

In terms of this research, co-operative authoring and collaboration are achieved by using the Web as the interface between the user and the system. This is achieved initially by using the ability of web pages to display forms for the input of data. These forms are used to display the current form of the web page in pure HTML, and they allow the user to edit this and then to submit the changes to the web server. Although this method of editing the raw HTML makes it more difficult for the casual editor, those that know HTML should be able to do so with little trouble. Although it could have been possible to create a subset of HTML to use, and format it appropriately, one of the ideas behind this thesis is to allow anyone to edit the pages, as full web pages. This means that there are no artificial limits placed on the content of the web pages (other than for security, by removing potentially malicious code). No artificial limits means that anything that is available for insertion on web pages is available for use in this system, rather than limiting what can be included to any particular subset or variant of HTML. In the future other methods of editing the web pages will be available, such as using many web browsers' in-built editor. This would be supported using the "PUT" operation that is available in HTTP (HyperText Transfer Protocol), which allows uploading of contents to a web server. Other options include dedicated web page editors, other methods like WebDAV, and other non-web access methods.

One of the reasons why this research should look at co-operative authoring and collaboration is that it allows the Web to be used as more than an information distribution medium. It allows for a free flowing web of information, in all directions, person to person (through the web site), person to web site operator, etc, rather than the normal web site creator to web surfer. This should enhance the way the Web is currently used and allow greater use of the Web for collaboration. It also has advantages that allow for more than just use over the Web, and so should be able to be used for any co-operative authoring and collaboration.

3.2 Co-operative authoring and the Web

Since the Web was created in 1989, it has been growing at an astounding rate (the beginning of the trend is detailed in the Post Web creation history (1989-1995) timeline displayed earlier in section 2.1). It is becoming (or already has become) an essential part of many people's lives as a lot of people either have access to the Internet, or will have access in the near future through a variety of devices (such as computers, TVs, cell phones, etc). It is important to use a technology such as the Web that allows a large portion of the potential users to be able to access it, and to communicate with one another. Communication between people is essential for collaboration between them to take place.

A lot of people assume that the Web is all there is to the Internet, because the Web gets so much attention these days. In reality, the Web is just one of the more visible applications that run over the Internet, along with a couple of others such as Email.

Technically the application that is being proposed will run over the Internet, rather than just as an application on the Web. The reason that this research is starting out with the option to create it as an application on the Web is because by using the Web the majority of people will have quick and easy access to it. There is the potential to have more powerful versions that will work beside the Web version and support more advanced features. However they will require some effort on the user's part, whereas the Web version will be wholly server-side because it means that there is practically no installation or set-up required by the user, which will allow it to be readily available to a large number of people. However, the proposed system is not inherently Web-oriented. Later versions, running a more specialised protocol over the Internet could be more powerful, though at the cost of client-side complexity because they will require that the user installs an application on their computer. The reason that it will be more complex is because the application will need to be installed on all the computers that they use to access the system, and they may not be comfortable (or allowed to) install new software on those computers.

3.2.1 The future of the Web

The original vision of the Web was for a much more interactive medium than we currently have. Currently there is a reasonably good system for one-way communication over the Web, and limited two-way (usually limited to email responses or specific local responses). This research is looking at what can be done to enhance the current lack of widely usable co-operative authoring available through the Web.

One direction that could be taken would be to follow current tends, and extrapolate from there. This would involve using the technology that we have at present, and only going from the point that we are currently at with regard to the possible future for the Web, without looking at what has and has not worked in the past. Thus we would not be looking back at the past to see what has and has not worked previously.

"Those who cannot remember the past, are condemned to repeat it." - George Santayana [Santayana 1905].

If we use the current technologies, but at the same time look back to the past to see what has and has not been done, we are much more likely to avoid repeating the past that we wish to avoid, and more likely to repeat successes that have happened in the past. This should be able to be achieved by using current technologies, such as web browsers and editors, and following the trends that made the Web successful in the first place, such as integration of existing data, open or public standards, being cross platform, etc. We could take the current state of the art, both of web browsers/editors and other document-editing applications, and make sure that we take into consideration the original intentions and ideals, for example the collaboration and ease of access intentions, while adding the best of new ideas. By using and pushing the limits of our current technology, while we make certain that everyone can use it, we should be able to lead the Web to a long and fruitful future that surpasses our current expectations.

How the Web became successful

To take a look at how the Web became the success that it is today, one needs to look back at the traits that the Web had at the time, to see what possibilities worked. By looking back at how it became successful, there are some traits that would be useful to follow for any future endeavours, such as this research.

The success of the Web has come from some basic traits that have allowed it to expand at a tremendous rate, such as:

- Having a core initial user group (at CERN)
- Integration of existing data such as diverse databases and systems that required specific applications to access.
- Using standards that were either already in use, or that were easily used (both simple and public, or open)
- Being cross platform
- Working across organisations

Success on the Web today

Just looking back at what made the Web successful to begin with is not enough. To be successful with any new endeavour, one must also know what is successful today. This allows the original reasons for success to be combined with the current ways that the Web succeeds in order to emulate those successes.

The main area that is a resounding success on the Web is the dissemination of information. One of the main problems with this is that the sheer amount of information that is available can at times be a problem. There were 21,166,912 web hosts when Netcraft conducted a web server survey in September 2000 [Netcraft 2000]. This can present a problem to people who are searching for information, as it can sometimes be difficult to locate the information that you require, even when it is out there.

Other areas that are quite successful, in a well designed web site, include:

- Using the Web for general reference material (provided that appropriate precautions are taken for screening out inappropriate material, such as XXX or sex sites, from minors)
- Support mechanisms for collaborative design of other things (e.g. many Open Source [Comerford 1999], [Hecker 1999] programs. Open Source is a software development model that harnesses the many programmers spread throughout the Internet, rather than the traditional model of harnessing a few programmers in a specific location. See http://www.opensource.org/ for more information).

- User support (The ability for users to search for and find information that will help them with any queries or problems)
- Sharing information that you possess
- Allowing universal readership (although it is a pity that many webmasters don't take note of this one. e.g. a site that requires graphics, or scripting)
- Format negotiation - the ability to receive information in the most appropriate format (such as your native language, provided the web site has a translation available).
- Searching, provided that you know what you are searching for, and how to express it in a non-ambiguous way (not always as easy as it sounds)

Some common misconceptions about collaboration and conferencing

There are some common misconceptions about what makes an application a group collaboration, or conferencing tool. Common misconceptions about these tools are listed below. As can bee seen by some of the misconceptions, it may not be clear to people just what a group collaboration tool is, and what a conferencing tool is, and they do have some overlap, which can add to the confusion. The reason for mentioning conferencing is because some people are not sure if there are any differences between conferencing, and collaboration. Collaboration can include conferencing as part of the collaboration process, just as co-operative authoring can be part of the collaboration process, but conferencing and co-operative authoring are different. Conferencing is more for getting a group of people together at the same time to discuss, whether it is a conference in person, teleconferenced, video-conferenced, or network-conferenced (with text only, sound only, or with video and sound, and possibly shared screens and the like). Co-operative authoring on the other hand is creating documents by a group of people that are not constrained by time or place, and can work on the documents with little change to their normal methods of writing.

Some common misconceptions about web collaboration and conferencing are:

- that they only refer to threaded discussions on web pages, or real time white-boards
- that they only entail group discussions
- that they are not real time
- that they are real time
- that they must use a web browser (even if it is not the natural way of communicating that information)
- that they must be text messages
- that they must be full multi-media

Original and current Web collaboration

If you take a very weak definition of collaboration (in that most of the information flow is one way, with just comments and the like coming back) then the Web is already a very successful collaborative environment. If you take a view that is more in the spirit of W3 with a slightly tighter perspective of collaboration, then the Web has some way to go from its current domain of information publication, to a free flowing web of information.

One of the main concepts in the creation of the Web was collaboration [Berners-Lee 1990]. As detailed above, the Web has had major growth, but opportunities for collaboration have not been seen to be utilised. If groups do exist that use the full potential of the Web, then they seem to be in isolated pockets with their focus inwards, such that those that use it know about it, but no one else knows. If this is the case, then these groups do not seem to be providing much to enhance the potential for the Web for everyone else. The Web does allow collaboration that was not possible before, but much more can be achieved. Previous attempts at collaboration (except maybe for Berners-Lee's original ideas, such as the original 'WorldWideWeb' browser/editor for NeXT [Berners-Lee WtW, Chap 4]) have tended to use collaboration that is already available in some current easy form. The main two at present are real time chat and video calls, that are both based on the concept of the original 'Talk', IRC [RFC2810-3] and MUD applications/protocols, protocols that are still widely used, and on converting Usenet/Mailing lists to web form. When set up correctly, the Web does a good job of archiving Usenet and Mailing lists. In the current form, using the Web for access to Usenet or Mailing lists is very "clunky", and difficult compared to using email and Usenet in the first place. One of the reasons why the Web is not as quick and easy to use as the normal programs is because every command must be returned to the web server to have any effect. This has a great effect on slow connections like dial-up, and still has an effect on fast connections. Because all the information is stored only on the web server, it is not possible to download the new email/newsgroups and read them offline. Also you are limited to the actions that they have implemented for the web interface, rather than the variety of options that are available through the range of email and Usenet programs. When used for comments on articles, the Web does better, but can still be improved on. Two examples of web sites that allow comments on articles that are posted are Slashdot (http://slashdot.org/) and MSNBC (http://www.msnbc.com/). Email and Usenet should be used the way they were intended and then the results can be archived to the Web for storage and searching. This gives the advantages of both, without many of the disadvantages of either. These points also affect the YEdit engine, which is one of the reasons that Web user interface will not be the only user interface.

A new protocol called WebDAV [RFC2518] (an open distributed authoring and versioning protocol) supporting HTTP [RFC2616] extensions for distributed authoring will help collaboration efforts over the Web, as it adds extra abilities to the HTTP protocol that will help with collaboration, versioning and related actions. This new protocol will fit in with this research well, as this research can be extended to work with the new protocol, and that will ease actions like locking, versioning and transfers of files.

Standards should be employed wherever possible, unless there is a compelling reason to use something else, as they are generally easier to use than methods that just seek to

replicate the standards. Also by utilising standards, compatibility and interoperability are enhanced. Standards such as HTTP access will allow any standards-compatible web browser/editor to access, author and publish information over the Web. In the future more advanced methods will be used, such as the WebDAV protocol, and/or other standard protocols for non-web access.

Areas that the Web has not been as successful

Today, if you look at the success of the Web, it would be easy to assume that the Web is a complete success. If on the other hand, you take a more in-depth look at the original vision for the Web, part of which involves collaboration, a different picture emerges.

The collaborative features contained in the original intentions for the Web form part of the basis of the idea of a web through which information flows freely in all directions. This allows for information to flow not only from web sites to users, but also from web site to web site, user to web site, and user to user (through the web site). This is achieved by allowing the web site itself to be the medium for the communication of ideas. This is quite different from the main information distribution model that is currently seen on the Web, that of the web site being controlled by one person or group, with them providing all the content, other than limited areas such as web boards, comments and such.

The collaborative features of the Web have not been as great a success as other aspects of the Web. If you don't look back in history, to both the original design goals for the Web, and the other visionaries that envisioned a future with free and easy access to information in a easy and connected manner, it would be very easy to get a slanted view of whether the Web has lived up to its full potential. There has been some work in the collaboration area with CSCW and BSCW, although these focus on the processes that surround the creation of a document, rather than the creation of the document itself. So for example, they have the ability to track external notes, dates, agendas, but the document itself is generally in some other format and is processed in a file-sharing manner, similar to FTP, file shares, or email transfer.

The free flowing Web of Information

The goals of implementing this research (Co-operative authoring and collaboration) are to provide options for co-operative authoring and collaboration over the Web that will enhance the potential of the Web. This will allow much more information sharing than is currently easily available, and would allow enhancement of the Web in the areas that it is currently not fulfilling its potential. This collaboration would include the documents that are being created, along with the collaboration surrounding that work. As the web pages could have the ability to be edited by anyone (or a set group), a larger number of people would be able to provide information, especially those that may just have a few words to contribute, which would not be worth setting up a full web site for. This would also allow people to work in the environment where they feel most comfortable, and would allow them to be more productive. This environment might not be a computer, as we currently know it.

3.3 Collaboration today

Currently there are many different forms of collaboration and co-operative authoring, and all of the computer-based forms of collaboration have a basis in manual pen/pencil and paper-based collaboration. There are some products such as Microsoft Word 2000 that provide some web integration within the word processor, the problem with this though is that most of the useful features are limited to working with their own products. I have yet to see any that go beyond simple group collaboration (with the occasional ability to publish limited sets of information automatically), and make it possible to work fully on the Web, without requiring specific proprietary programs installed on the computers accessing the information.

Currently there are two common methods for creation of a group document (word processor file, source code, etc). Other methods of group document creation will be looked at shortly. The first common method is to write a portion of the document and pass it along by email (this requires everyone to have the email addresses of everyone else). Each person adds to the document when they have it, or they make changes to an original document and then an editor combines all the changes. The second common method is to use a shared directory where everyone in the authoring group can access to edit the document. This requires everyone who is to collaborate on the document to use the same program to edit the document, and requires them to turn on the change tracking mechanism.

One problem with this approach is that each of the applications used to edit the document must be capable of reading and writing the file formats produced by all of the other applications. Even if all the authors use the same editing application, different versions of the same program can cause incompatibility problems. Often problems such as these only show up after continued use.

3.3.1 Collaborative editing and version tracking packages

Some packages have version tracking mechanisms, but the user can be limited to using one program, (sometimes others may be compatible, but unless they are completely compatible, problems may not show up until it is too late to solve them). Generally different versions of the same program will have compatible versioning information, for example Microsoft Word 6 and 97.

For work in an environment that enforces the use of only a particular software package, then it may be possible to just use the in-built abilities of the particular package.

On the other hand, when an organisation needs to collaborate with people outside of that organisation, or they allow their staff to use different applications for the same purpose, then there are more factors that need to be taken into consideration. The people that the organisation wants to collaborate with may not have a package that supports collaboration in the same way as their package, and even if it does, there can be problems when information is saved in different incompatible formats. This could cause delays in the collaboration and cause major problems if there are time sensitive issues associated with the collaboration.

3.3.2 Version tracking information

It is tempting when dealing with the problem of what to do when people need to work with others who work in places using different packages, or work with others outside the organisation, to avoid any question of version tracking. This is achieved by keeping very little in the way of version information; maybe a copy each time a change is made, assuming of course that there are backup copies available of the latest version at all times. Often this may be enough information, but in a complex document someone may need to know who changed what and when, or what changes a particular person made, or other specific change information.

If all that is needed in a package is the ability to save a version occasionally as a backup (for example, a copy of the document as at the end of each week), then any package will work much the same as any other. Unfortunately this will result in the loss of information about who changed what, although when the change was made can be narrowed down by the length of time between the backups.

Simple backups may not be enough if the document being created is large, complex or involves many people.

3.3.3 Currently available options

For documents such as source code, commonly used programs have a 'check in and out' facility to allow for versioning. This stores each change in the file along with information on when it was made, and by whom. This works well for those who understand the complex issues and programs that they have to drive. Some examples of these programs are CVS [CVS], [Fogel 2000], PRCS [PRCS], RCS [RCS], and SourceSafe [SourceSafe]. These can be quite powerful and complex programs to use to control source code and the user interfaces are improving all the time, making it easier for people to use them.

CVS (Concurrent Versions System) [CVS], [Fogel 2000] is an example of a very commonly used source code revision-tracking tool. CVS is a "Source control" or "Revision control" tool to keep track of changes to source code files made by multiple developers. This allows multiple developers to work on the same source code at the same time. CVS keeps a history of all the changes made to a set of files in one central location, so that multiple developers can check in and out the code that they are working on. Another ability is to be able to branch versions, so that there can be multiple branches for different objectives, for example one branch for vendor code, one for the current version's bug fixes, and one for the next version. Because all the history of the files is kept, any changes that cause problems can be compared against an earlier version that may not have had those problems. CVS also has the ability to automatically merge files that have been changed by multiple people at the same time, as long as the changes have not been made to the same location in the same file. If changes have been made in the same location in a file, that file needs to be manually merged. CVS is also available on a variety of computing platforms, and can be used in a client/server mode to allow developers to access it from any location.

There are other version tracking tools. Another of the tools is RCS (Revision Control System) [RCS], which is used by CVS to do all of its underlying work, as RCS is for

tracking individual files, rather than multiple files like CVS. PRCS (Project Revision Control System) [PRCS] is a similar revision control system to CVS but it lacks multi-platform support, and currently does not support a distributed client/server model. Another version control system is SourceSafe [SourceSafe], which is primarily aimed at only the Windows platform, although there are a couple of other companies that provide compatible programs for other platforms.

Each of the major word processing suites has some version tracking included that allows some group collaboration. The version tracking that is included in these suites is generally simple tracking that tracks the changes that have been made in a single document, and stores who made the change and when. This tracking ability is limited when compared with the abilities of the above group of source code revision systems. Examples of these word processing suites are Microsoft Office (Microsoft http://www.microsoft.com/office/), WordPerfect Office Suite (Corel http://www.officecommunity.com/), and StarOffice (Sun http://www.sun.com/products/staroffice/).

In the process of creating other approaches to collaboration some try to only replicate the features of an earlier technology (such as email or Usenet), without looking at the original inspiration and current implementations for that technology. In the process, they sometimes implement features that are not really needed, and don't implement the features that really are required. As an example, the web based BBS (web based bulletin board systems, not to be confused with the dial-in BBS's that were a lot more popular before the Internet took over) try to replicate Email (SMTP [RFC821 / STD 10] and POP3 [RFC1939 / STD53]) and Usenet (NNTP [RFC977/RFC1036]). Web based bulletin board systems typically provide web pages that allow you to add a comment to a web page. This can sometimes be simply added to the end of the page, or in more sophisticated systems, they can be threaded in a tree like structure. Often they are trying to achieve a discussion forum by trying to replicate Usenet on a web page. Usenet is the discussion forum that has been around since a discussion forum was needed on the early Internet. It provides a place where messages can be publicly posted and replied to, which has been in use for many years (since 1979). Email and Usenet model the conventional mail and notice board systems online, on the Internet. The web based BBS's try to constrain Email and Usenet to a web based interface. These web-based interfaces do not contain the features and benefits that many people come to expect from an Email or Usenet. For example the offline use, that is the ability to download the messages, and then read and reply to them at your leisure, which also cuts down on the cost, as many places have to pay for an Internet connection by the minute. The ease of use of a program that is dedicated to reading Usenet or Email can be a great advantage as it was designed to work with Usenet or Email, rather than just being an add on. Finally the speed of use is usually much faster because the programs usually have advanced features for reading and writing, and they are dedicated to these functions, things such as downloading messages to read, rather than having to get each one separately when you want to read it.

Following are some examples of how technology has impacted on the methods used for collaboration

Face to face meetings

Face to face meetings, both formal and informal, are an important part of life, whether it is a one-on-one, for example talking things over with a work mate, or a group meeting such as a presentation, a brainstorming session, or another type of meeting.

These are important for sharing information and communication about projects and the like. They are also important for collaboration, as it allows you to talk with others about both your ideas and theirs, and how the ideas interact with each other.

Video/Phone conferencing

Video and phone links allow you to talk with people in an office down the hall, or on the other side of the world.

This has the advantage that you are not constrained to only talking with people who are at your present location, but it has the disadvantage that you lose out on a lot of the non-visual clues (video conferencing helps, but is still limited).

Instant messaging / Internet chat (IRC, Talk, ICQ, etc)

Instant messaging and Internet chat (such as Talk, IRC, ICQ, etc) have the advantage over video and phone connections that they can be used to connect many people together instantly, at low cost, while they are using their computers. Some of these can store messages so that if the recipient is not at their computer at the time, they can pick up the message later. You can also be talking to a number of people at once, and not all of the conversations need to be accessible to the others that you are talking with. Although the ease of use is much higher than face-to-face meetings and video/phone based communications, especially for communicating with people who are not in the same location, there are some disadvantages over them. Not only do you lose the visual clues as to what the person is saying, you also lose the audio clues, the inflections in the voice.

Interactive Chat

Other options are interactive authoring of documents with whiteboard / chat / screen sharing tools. The major disadvantages of tools such as these are that all the people who are to work on a document must be co-ordinated to connect at the same time using the same software, and then they all work on the same document at the same time. This can be useful for demonstrations or for brainstorming, but it can be tricky to actually write content for the document in a session such as this, unless there are just a few people.

Email, Usenet, mailing list

Email and Usenet are similar, in that they are both intended for sending text messages from one person to another. Email is primarily for one to one (or a few), and Usenet is primarily for one (or many) to many. There is some middle ground, where there is a group of people who want to talk with each other, while not opening it up to everyone, and they commonly use either individual Usenet news servers, or mailing lists that contain the people that they want to talk with. These methods add to the ease of use of the Internet chat, but they are designed for conversations over time, not for real time chatting, although at times this type of communication can get very close to this. This has the advantage of not requiring everyone to be present when you communicate with them; they will get the information when they next check their email or news. This is useful for general communication, but has the same limitations as instant messaging and Internet chat, in relation to not having any visual or audio clues to work from.

Word processed documents marked up electronically or via printout

Often people will create a document in a word processor which can be set up so that people can see the alterations that others have made to a shared document, and then pass that electronic document on to someone else to work on. This can be done either using the inbuilt methods for tracking changes that some word processors have (such as using underline for additions, strikethrough for removals, and different colours for different authors), or it can be achieved by printing out the document and then marking it up manually. Even when marked up manually on a print out, this still has the advantage that the original electronic version can be easily edited. When changes are made electronically the editor can either accept or reject the changes to the document with much more ease, but it requires that all users use the same word processor. These are in stark contrast to the completely manual method discussed next that requires that the document be completely rewritten/re-typed for every change (as paper does not have the ability to easily change text, move it around, add text, or remove it).

Typewriter/Hand-written manual mark-up

Collaboration via typewritten or hand-written documents (in other words, **no** electronic version to re-edit) is very much in the past in most places today. If you look at old documents, especially if you can get your hands on documents that were in the process of being created, you will have the chance to see how collaboration on documents was achieved back in the days before computers were readily available. The document would be created and then passed on to the reviewers, collaborators, etc, and each of them would have a coloured pencil or pen. They would then edit the document, using the coloured marks to show who changed what and when they changed it. One of the major advantages that this has over (even current day collaboration) is that anyone could view the document, anyone could edit it, and it was easy to keep a record of those changes, a physical paper trail. The major disadvantage with this method was that each time changes were made, the document had to be rewritten, and that you had to have physical access to the document to make any changes. This meant that it could take a while to post the document to someone to edit, and then to post it back, and repeat.

This has had a major influence on my ideas for co-operative authoring and collaboration, and this is one of the reasons that the application is not limited to one front or back-end. The approach taken in this research is to use the good ideas that have been used in the past, such as the ideas from hand/type-written paper-based collaboration and editing, and to move along similar lines. It will be usable over the Web, locally, or using other access methods. This mirrors the flexibility of paper, in that it can be viewed by anyone, at any time, anywhere, but it will go beyond the limits of paper such as the limits of physical transfer (both of the information and the transport of that information), reproduction and will enhance the possibilities of use. It also demonstrates for those that may not realise, that co-operative authoring, editing, and collaboration are a part of our history, and by looking back at the systems that were used for paper, it is possible to use some of the properties of this medium to enhance the Web based version.

3.3.4 Web collaboration today

The major problem with the majority of these tools is that they are limited to people who share the same working environment, so people are forced to use software that they are not familiar with, and less productive with. Some of the tools only work interactively, (which works for meetings and demonstrations, but means that everyone has to be organised to a specific time) for any work to be done. There are some experiments underway with regard to co-operative authoring, such as WikiWiki, and web collaboration such as BSCW.

WikiWiki

WikiWiki [Cunningham] (http://c2.com/cgi-bin/wiki) is a web site created by W. Cunningham that allows anyone to edit the pages, with certain restrictions. This is the first Wiki; others have appeared, as people have set up their own Wiki clones. WikiWiki was written in Perl [Dominus 1998], and some of the newer Wiki clones have been written in other languages, but they all follow the same general guidelines. A couple of the general guidelines that the Wiki clones follow are that anyone can edit and add pages, that they use a simplified mark-up language, rather than the full HTML standard. As the Wiki clones evolve, these guidelines are subject to change as the majority of users of Wiki decide.

WikiWiki can be described in many ways, such as this quote from its home page (current October 2000) "(It is) ... a fun way of communicating asynchronously across the network", or as a set of web pages that are open and free for anyone to edit as they wish. Wiki is not real-time; therefore people have time to think before they follow up a web page, often days or weeks, so they have time to consider what they write. It was created for the discussion of People, Projects, and Patterns (a pattern is a recurring solution to a common problem in a given context and system of forces [Alexander 1977][Alexander 1979]).

BSCW

Quote from BSCW's [BSCW] web page at http://bscw.gmd.de/

> *"BSCW (Basic Support for Cooperative Work) enables collaboration over the Web. BSCW is a 'shared workspace' system which supports document upload, event notification, group management and much more. To access a workspace you only need a standard Web browser."*

BSCW supports CSCW (Computer Supported Cooperative Work) over the Web. [Bentley Horstmann Trevor 1997], [Appelt 1999] This increases the reach of CSCW, as it allows anyone with a web browser to access the information.

As the titles for both BSCW and CSCW imply, they provide support for co-operative work. When looked at beside the free flowing web of information that I am proposing, they are complementary, as the free flowing web of information is concerned with the co-operative creation and editing of documents, whereas BSCW and CSCW provide support for that co-operative creation and editing. The way that BSCW and CSCW work is that they are a support mechanism for co-operative work (which is what the "SCW" in their titles stands for). The support that they provide varies depending on the implementation, but they generally support the work around a document, rather than the work specifically on the document, often this is left to the tools (word processors and the like) that the members of the group have.

What is needed is a form of co-operative authoring and collaboration that anyone can use and access at anytime from anywhere using the system that they are most familiar with and productive in. The other advances such as WebDAV (which is discussed later in the next chapter), and BSCW are important, as they provide the support for areas that are not directly covered by the free flowing web of information, in that they specify the protocol to be used over the Web, and support surrounding the co-operative authoring respectively.

3.3.5 Lost Versions

The free flowing web of information can also solve the problems of old/new documents and of revisions (both intended and unintended) causing lost information

Thankfully the Web Consortium (http://www.w3.org/) has kept historical information about the development of the Web, and left some of the web pages as they were written, and they have refrained from updating them as new information has become available (which means that all the historical information is still there). This is important because one overlooked problem in updating documents and keeping them current is that you can unintentionally lose valuable and important information. This commonly happens when documents are rewritten to make them clearer (as the original emphasis can easily be lost), and seemingly unimportant information can be dropped, as it just seems to add clutter to the document. Old information may also be replaced in a document by new information, which can be very good for the immediate purposes of the document, but the information can be permanently lost if the document is its only repository.

All of the above, along with restructures (either company, web site, or other), can easily and quickly change documents, unless a full history of the document (not just changes) is kept somewhere. When this happens, a lot of good information can be lost until somebody else comes along and reinvents the same things. Often they do not realising that there was a good information base available in the past that would greatly ease any trouble they have reinventing it. Occasionally when this seemingly irrelevant old information is stumbled on by somebody who is interested in that particular area (especially when they are looking for the old, original documents, ideas and objectives), the results can be significant, once the information has been processed into a form that they want.

This brings to light another point that is important when it comes to old documents (not only web, but also other electronic forms, and even hardcopy).

Information may not continue to be available over time. A document that was written yesterday may be easily available, but then again it may not. There are many causes of information becoming unavailable, and this can happen at any time, potentially even immediately after the information has become available. For example a document may be created, the only copy (in other words, no backup) might be stored on a floppy disk, or a web site, and then the floppy disk becomes damaged, or the web site crashes, losing the information. There are many other potential ways for information to be lost, even within hours of it being created or made available, and a couple of the longer term access problems will be discussed next. That is the information is still there, but there is no way to access that information.

With the current rate of change in the computer industry, and the associated changes in storage formats, documents that we now rely on may not remain accessible. As an example, if you were to try to access information that was stored on a 8-inch CP/M floppy, or tried to read an old word-processing format, you may find that they are no longer supported, and you may not be able to retrieve the information.

Of course incompatibility between old document formats and new applications is not the only side to the continuity-of-information coin. Another problem that occurs is when a document has been created in the latest version of a program; a previous version may not be able to read the file. This happens when people with an older or reasonably current word processor/web browser try to read something that was created in the latest version, and saved with no thought about those with current or older software/hardware, or even those that don't have everything enabled by default. The best case is that the person (who in the case of web pages is surfing the Web) that is trying to access the information realises that something is wrong, and they try to do something about it. If it is too much trouble, they may not bother (which occurs all too often), or even worse, they may see nothing, and so assume that there is nothing there, and they never come back.

One example of this problem is when a someone using a particular version of a word processor, suddenly finds that they can not read a document, because one of the members upgraded their version of the word processor to a later version. A second example is when you visit a web site of a company, and all you get is a blank page (because they assume that you have images and/or JavaScript/VBScript turned on). Some people have images turned off because they take a long time to download over a modem. Also some people do not have JavaScript/VBScript turned on, as it increases

the complexity of rendering the web page in the browser and has the possibility of leading to some instability.

This demonstrates the need to keep old versions of documents, even versions that seem at the time to have no significance. It shows the importance of keeping at least all of the major revisions of documents, in addition to at least the most recent set of changes so that any changes that have unforeseen consequences can be rolled back and any problems solved with ease.

3.4 Types of Web publishing

Creating the web sites as multi-tiered, that is, having different areas (that may be located in different places) that support the main activity of the user, should ease the use and stability of the web sites that implement the free flowing web of information. Showing the full co-operative authoring interface to all users is probably not ideal, as it would increase the complexity for both first time users, and those that only want to read the web site. The advantage of a multi-tier web site is that it can combine different types of web pages in a layered fashion. That is, the pages can be dynamically created, then stored as static pages, which allows people to access either set of pages, depending on the functionality that they want. For example a web site that allows pages to be edited could have the dynamic site available for both read and write access, while keeping an up to date set of static web pages, which most people access for reading only. This allows people read access at all times, even when the dynamic site is having problems. This is applicable to all web sites that use dynamic generation for web pages and can be very useful to ensure reliability. This was not included in earlier versions of the YEdit system, but is now included and can be used with little extra set up required. The differences between the different types of web page are discussed below.

Static web pages

There are several different types of web pages. The easiest and probably the most common is the static web page (a web page that is written and then served to the user as is, with no other information added to it), all the web server does is display it. Often these types of pages will be the basis for an interactive site, as static pages load faster, and because they don't have any changing information in them, you know how they will be displayed, as there is less to go wrong in them.

- Advantages of static web pages are that they are fast, reliable, simple and easy to create.

- Disadvantages of static web pages are that they can be time consuming to update, and it is easy to have broken links and associated problems because a web page or link was missed when they were updated.

Static web pages with Dynamic content

This type of web page is written as a static page, but has statements in the page that the web server will execute. Examples of this type of page include pages that call SSI (Server Side Include) functions, embedded Java Servlets, and a few others. These are commonly used to include dynamic information into an otherwise static web page. This can also include having the static web page as a template, and using the dynamic content to fill that template. There are many options and different ways of combining static web pages with dynamic content. There is a risk that if the dynamic content is not available when the web page is requested, that the page may not display properly, or at all.

- Advantages of static web pages with dynamic content are that they are much more flexible than a static web page, it is possible to update the content without changing the web page, and they are somewhat customisable to visitors.

- Disadvantages of static web pages with dynamic content are that some web pages may be slow to load, while some are fast. This variation in behaviour makes the web site seem inconsistent, so the web surfer does not know what to expect next. The dynamic content can potentially have quite complex interactions, which can cause problems for both the web master and the web surfer. Also static web pages with dynamic content are less flexible than a fully dynamic page.

Dynamic web pages

Dynamic web pages are fully generated on the web server commonly using the CGI (Common Gateway Interface). CGI is the main method of calling executable programs from the web server, and it allows you to call almost any type of executable. Often the executables are written in Perl (a script language), an executable, or a shell script (such as 'bash' on Unix or batch files on Windows) and most are located in a "/cgi-bin/" directory on the web site. There are exceptions to this; for example some CGI programs are run from individual directories, and some are called via other means, such as Java Servlets that are commonly located in the '/servlets/' directory. Of course they can all be embedded in a web page for use in a static web page with dynamic content, or located elsewhere on the server, provided that the server is set up to expect that. With dynamic web pages there is an even greater risk that pages will not display properly, or at all, if there are any problems with the programs retrieving the content, or the content itself.

- Advantages of dynamic web pages are that all information (including the web page itself) can be updated in real time, and the whole site can be custom-adjusted to visitors.

- Disadvantages of dynamic web pages are that they are slow (compared to static pages), and so require a more powerful web server. They are less reliable than a static web page, as there are more places that a problem can occur, and they are much more complex than static web pages (even ones with dynamic content).

Of course in the real world, there are many pages that straddle the boundaries between those definitions. One example is that a web site might have an index that is updated once a day, but is stored in a static file for the rest of the day, and could be served either straight from the static file, or through a dynamic page.

Multi-tiered web sites

A multi-tiered web site is one in which different parts of the site are located in different places. This can be as simple as having static pages on one site, and dynamic pages on another, through to having separate web servers each with a specific function. For example, there may be some to serve static pages, some to serve images, some to serve the dynamic pages, and some to store the dynamic information (commonly in a database).

One of the major advantages that a multi-tier web site has over one that combines everything is that any problems that occur should only effect one portion of the web site. A multi-tier web site commonly has the best advantages of each type of web page, while reducing the risk that a problem with any one part of the system will bring the whole lot crashing down.

The advantages of static pages are their speed and reliability, and the advantage of dynamic pages is their flexibility. By combining them using a multi-tier web site, the advantages of both can be combined, such as the reliability of static pages and the flexibility of dynamic pages. Also the disadvantages of each should be mitigated to some extent (that is, the lack of flexibly of static pages, and the speed and reliability of dynamic pages). Unlike sites that combine static web pages with dynamic content, the multi-tier web sites split the two by placing them on different servers or at different locations, so if one fails, the other can still keep serving web pages. They would not be quite as reliable as static pages, or quite as flexible as dynamic pages, but they would be a lot more useful. This view is similar to caching the output of the dynamic pages and then using that for all other requests for the page, until the dynamic page is updated, that then automatically updates the cached page. The difference from caching is simply that the dynamic page updates a static page (that need not be on the same server), and the web server then serves this static page (that was written by the dynamic page) as if it were a static page.

This splits the dynamic content pages from the static pages, and would allow all the static pages to be on one web server, which is available for anyone to read, while the dynamic web server is available for those who are updating the documents. This means that even if the web server, which controls the dynamic pages, crashes; the only ones that will know (or possibly notice) are the people who are co-authoring the documents. Even then, they can still read the documents. This means that the web site is much more robust from the general surfer's point of view, and also means that the web master of the site has less to worry about.

One of the advantages of a separate multi-tiered web site (which has static read-only copies of the web pages on one server, and same documents, but editable on another) is that it gives the advantages of a static web site (that is the speed, reliability and simplicity), with out the disadvantages, by having the static site automatically updated, as is required, from the interactive editable site.

Disadvantages of a separate multi-tiered web site are that it is a little more complex to set up, and for proper redundancy you need at least two servers, preferably in different physical locations, with different bandwidth providers. If a less robust system is acceptable, then it could be set up to work from the same location, or if really necessary, on the same web server. This would be much less robust, but would still give better robustness than no multi-tier arrangement.

3.5 The future of the Web

How to make co-operative authoring successful

Some ways of ensuring the success of co-operative authoring over the Web are to require that people can use the programs that they are currently used to using and to make everything as easy to use as is possible, as well as incorporating some of the methods that made the Internet and the Web as successful as they have been.

Some of the ways to achieve these goals are to use tools on the server for as much of the work as possible, because they allow the user to do their work wherever they happen to be, even if they have not got their favourite programs with them. Other ways to achieve the goals are to limit any requirement for client side tools to those that need little or no user intervention to work (such as Java applets in browsers). Also to take advantage of the programs that the user already has (such as web browsers), rather than requiring them to get more programs to clutter up their computer. They must be cross platform, Unix, Windows, and Mac, as the bare minimum. Tools that are already available should be used (don't reinvent the wheel). Examples of such tools are online chat (IRC, Talk, ICQ, etc), email, Usenet. Enhance them with **new** abilities, where they don't already exist, but don't replicate existing features, just to put them on the Web.

The future of the Web

Web sites of the future will be much more advanced than we are used to at the moment. As technology moves forward it will make it possible to do things that we can only dream about at the moment. One of the ways to enhance the Web is to bring about a resurgence of more of the original ideas that led to the creation of the Web in the first place. Ideas such as the free flowing web of information that allows co-operative authoring of documents, along with collaboration to work on those documents. The ideas of co-operative authoring have been around since even before the Web, and have been used to some extent, but the ideas that people have had in the past have not caught on, possible because other major things were happening at the time such as when Berners-Lee created the 'WorldWideWeb' browser/editor of NeXT.

By utilising ideas for new web sites with ideas for co-operative authoring and collaboration, we can combine them in new ways, with our current technology. In the future web sites will be as easy to create and to edit as conventional documents are to create and edit with word processors, and both types of information representation will be used as often as each other (and in conjunction with each other).

Web sites based on the multi-tier model above will allow webmasters to set up sites for people to use co-operative authoring and collaboration just as easily as they can create documents in word processors today. The collaboration will be available without added complexity and will allow a variety of clients to edit, update and view information on the site (with full version history, and security).

To achieve this, the web site will need to combine different technologies into one coherent bundle. The web sites will also need to be robust and well designed to keep the complexity of the process hidden from the user, and to present a simplified model of use. This will allow all web sites to incorporate the full potential of co-operatively authored web sites, while keeping the whole process simple enough so that anyone can use that potential. The site designers may not use all of the potential for every web site, but will be able to pick and choose the parts that are most appropriate for them.

A multi-tier web site such as this that has a dynamic back end (that may be on a different server, for reliability) that updates a static front end whenever a change is made on the back end. This combines the reliability and speed of a static site, with the flexibility and dynamism of a dynamic site, with the good points from each, while mitigating against the bad points. It gives the reliability and speed of a static web site because it is a static web site (that is updated periodically by the dynamic site), and the flexibility of a dynamic site, because the dynamic site can update the static site at any time (generally whenever something changes). So even if the dynamic site crashes, the static site will still be available and contain the information up to the point that the dynamic site crashed.

In the past few applications other than web browsers and editors could understand HTML, and therefore would not have been able to edit web sites (and in some cases, even though they can now read and write HTML, you would not want to use them to do so). Advances in applications and the protocols to access web servers are advancing such that many more applications are able to interact with web sites and support the standardised (W3) web features (in other words, no OS or browser specific HTML, that could break the display of the page on other systems), that may, or may not be a web browser as we currently recognise them. For advanced document creation with YEdit additional software or a reasonably recent web browser/editor (although this will still be platform-independent) may be required for extra features (although these features should at least be partially available for use in any web browser/editor). These extra features are only for advanced web page creation (in other words anyone can still view it easily).

All web sites that use the YEdit engine will have will have the ability to include full co-operative authoring and collaboration with no additional software required, on either end.

In addition to browser based, or advanced editing, the YEdit system will in the future allow other software to hook into the YEdit system, allowing the user to use the software that they are used to used to using. This will allow for features such as uploading and downloading of Word or WordPerfect files, for editing. Software should be the users choice; the server will convert to and from its internal representation as is needed. This means that the user will be able to use anything from EdLine (the infamous text file editor that was included with early versions of MS-DOS), through to the current web browsers and anything in the future to edit the documents with.

3.5.1 Increasing the success of the Web

There is a range of currently available technology (such as email and Usenet) that is being used for the purposes of communication, and quite often the same technology is reinvented again and again, sometimes using other techniques. Occasionally this brings out new abilities that were not available before; unfortunately, most of the time it only introduces added complexity to areas that can already be complex and difficult enough for the average user.

Letting people use the programs that they are used to and understand (rather than asking them to learn new programs often), will increase the ease with which people are able to use their computer to do what they want. It makes sense to let people use the programs that they understand because when they understand how to do things, they are much happier and productive (and a happy and productive user is MUCH less burden on a company). A user who is forced into upgrades, changes, and having to learn the software over and over again, when their current software works perfectly well for them, is often a burden on resources and support lines. On the other hand, if they are given a good reason to upgrade, they will, quite happily, with few problems; force them, and watch the support issues blossom.

3.6 One possible answer

The YEdit system (which a prototype has been created for, and is available for public use at http://www.YEdit.com/) for co-operative authoring and collaboration is based on the free flowing web of information that includes some of the original intentions for the Web; that is to use the Web for both reading and writing. YEdit will provide a possible answer to the problem of many people working on documents together that do not support easy co-operative authoring and collaboration, without extra work, such as remembering to email the latest version on to others.

In searching for the very information that I needed to work on this problem, I came up against one of the problems that I am trying to solve. That is, the problem that as information is revised, it is very easy to unintentionally lose important information. This problem is not so evident in print mediums, because once information is printed, it can be stored that way, and is very seldom rewritten and replaced. Unfortunately, on the Web, things are changing so quickly that information that is present one day may not be present the next. To keep pages updated authors often add, replace or remove information. Sometimes this is deliberate changing of information, but more often it is just that the information that was present, seems to the author to have less relevance now. The problem with this is that the very piece of information that one person may think is no longer relevant, and years out of date, may be the very information that someone is looking for, as I found in my search. Luckily W3C (The World Wide Web Consortium, http://www.w3.org/) did keep the old outdated pages, and some digging around their site revealed them, even if it took a little work.

While looking through these documents I realised that one of Berners-Lee's original visions for the web was along similar lines to what I was thinking would be a good direction for the Web to go. My idea was for an engine that could be used by almost any front end (be it web browser, word processor, web site, etc) that would allow interaction with full version tracking information. This would allow web sites (or any

server or application) to keep the full revision history for documents (any kind of document, text, pictures, sound, etc). It will also allow any groups that have permission, to edit, update and change any available document.

The engine that I am creating works as an extension of the web server. It can also work standalone for applications that require co-operative authoring, utilising interfaces other than the Web. The engine abstracts both the user interface and the file system interface, allowing other methods of interaction to be plugged in and work seamlessly. This allows for the engine to be used through a normal web browser and via other connection methods at the same time, using the same base of files. It also works the same the other way around, (refer to Figure 4: Interaction of the main components of YEdit for a graphical overview) allowing any base of files that it has an interface for, to be used by any of the access methods. This flexibility means that the engine has a much wider range of abilities to support co-operative authoring than just the Web, or any particular file system. For example it could be used via a command line interface, direct into word processors, and store information in databases, version controlled repositories, etc. This will also allow it to be used where legacy systems exist, either as the front end, or back end, or both, allowing disparate legacy systems to interact, when under normal circumstances they would not be able to.

The engine promotes co-operative authoring and collaboration by providing an interface that is built specifically for co-operative authoring, with full version histories of all documents, by default.

The Web interface to YEdit can be used to read different versions of documents in the same way as you normally read documents over the Web. This means that anyone can access the documents without needing to learn how to run a new application. The Web interface also supports modes for browsing the documents with co-operative authoring in mind (that is, displaying information that is appropriate to the user about the document, for example the version number, last author, last date of change, and previous versions). This part of the interface also allows you to edit the document, and at present has a simple locking scheme to prevent changes being overwritten. Because all of the changes are kept in different versions, it is possible to keep track of each change, who changed it and when. This allows certainty about the history of the document. The process of either reading the documents in the system, or browsing them with a view to editing them, or looking for information about changes, has been implemented so that as many of the abilities as possible are implicit in the display. Therefore anyone who knows how to surf the Web should be able to surf though the documents stored in this system, and need not know the inner workings to retrieve the information that they are looking for. The reading part of the interface is simply presenting the documents as they would appear on a normal web server, and may in fact be served as normal static documents, under a multi-tier web site design. Even the browsing part of the interface is designed so that the user is as unaware of the complexities of the system as is possible. This is achieved by providing a simple menu above the document, which provides information about the document, such as the last author, version number (and view versions), and the option to edit the document. Only if the user decides to edit the document does any of the complexity of the system show itself, and that will decrease as user suggestions (such as a simpler form for editing a page) is incorporated into the design.

This engine will allow web sites to be fully interactive, while keeping the version information that is important to ensure that the information is correct, and will even allow the whole site to be rolled back to a previous known working configuration, should disaster strike.

Because the full version history is included, and the ability to go back and check the previous versions, it will be easier to trust the information that you receive. If you have a query, you can easily check who changed what and when, especially if the site uses some form of write once media to either store the information, or to back up the new information.

4 Design through to testing

4.1 Design

The design approach chosen for developing the engine to drive "Co-operative authoring and collaboration over the Web" was the Spiral design model [Pressman 1992].

> *"A model of the software development process in which the constituent activities, typically requirements analysis, preliminary and detailed design, coding, integration, and testing, are performed iteratively until the software is complete."*
> *Source: IEEE Std. 610.12-1990.*

The YEdit system was created using the spiral design model, which involves incremental development, iterating through the cycle, starting with the ideas, design, through to the coding and testing. Each time through this cycle increases the functionality, usefulness and completeness of the application. Inevitably the application is refactored (Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure [Fowler 1999]) as it increases in size (this refactoring is an essential part of the process, otherwise the complexity would increase to a point that would not be manageable, and prior to that, it would become very difficult to continue).

This model has advantages over other models when designing applications in new areas. Unlike the Waterfall model, this model allows the high level design to change, as and when is needed, and also allows the design to flow both ways (that is, for discoveries during design, coding, and testing to be applied to the next round of the design), as new approaches are found to problems. As new ways of doing things are found, they are fed back into the loop as it goes around again. This allows new ideas to permeate back up through the design to the top level, and every so often these adjustments will lead to a large refactoring of the whole application. This refactoring decreases the complexity, and because the design has been flowing both up and down the application, a full refactoring is possible (and desirable) to do. In the Waterfall model, this would be much more difficult (generally the Waterfall model is best for applications for which you already have many of the building blocks). The Spiral model is useful if you need to create the building blocks, as you are creating the application.

The spiral approach is better for new designs where the emphasis is on the creative dimension of the development. The iterative approach with the Spiral model is all the more important when embarking on a novel type of project with a reasonable understanding of its overall design, but no clear idea of the details of its components. The iterative approach allows new ideas to be tried, and changes can be made in approach as new information comes to light. This is also important for the free flowing web of information because over the time that this has been created, new ideas about the system have surfaced (such as integration into word processors) and other methods of access have arisen (such as access via handheld computers, with much smaller screens, and access via WAP). The reason that an iterative approach is important is that the Internet is a fast moving area, and this design will need to be able to adapt readily to any new surroundings. The spiral model approach allows you to concentrate on the big issues first, solve them, and go through the process (which includes the full range of

software engineering, from ideas, design, right through to testing and feedback). At this point the whole process is repeated. Each iteration gives a finer resolution than the previous, and solutions can be found to unexpected problems. In fact, the three level design of the engine (user interface, middle interface, server interface) came from a refactoring of the code. This particular refactoring opened the way to splitting the user and server sides of the engine completely apart, as originally the user and server interfaces had a much higher integration.

Another reason that the spiral model was a good choice for this design was because the process that was undertaken started with just a broad idea. That was, to look at the Web and see how communication over the Web could be enhanced. In the process, several different areas were considered as possibly contributing to better communication over the Web, such as Web themes and multi-tier web sites. Possible ideas that could help communication over the Web were looked at in the early stages of this project, and one was incorporated into the final "Co-operative authoring and collaboration over the World Wide Web". These ideas were important to look at, as at that early stage, it was not entirely clear what area of improved communication over the Web would be the one that would be best suited to this thesis. The two main areas that were considered before settling on co-operative authoring and collaboration were web themes (similar to CSSs, but on the server side and more of a content/context split, rather than just enhanced display), and multi-tier/multi-homed web sites (to allow much more redundancy in case of problems), the main points of which were incorporated into this research.

Web Themes

Web themes apply styles/themes to web pages on the server side, allowing one web page to contain content, while using web themes to define the context. That is, all the content would be in one file, which simplifies the process of updating that information, while the web page contains pointers to the context information (like buttons, banners and other non-content information). These pointers retrieve the appropriate version of the context for the user, whether it is text only, or full graphics. Below (Figure 2: Example web page for Web themes) is an example of what the web page looks like on the server side. Each of the <SERVLET> code tags is replaced by the output of the servlet. The servlets retrieve the appropriate information from a database (part of which is repeated below in Figure 3: Example of the Web theme data), and return it as the output.

```
<HTML><HEAD> <TITLE>Web Themes Test sheet</TITLE> </HEAD>
<SERVLET CODE="theme"> <PARAM NAME="HTML" VALUE="Body"> <body>Your web server
has not been configured to support servlet tags. </SERVLET>
<a href="about/"><SERVLET CODE="theme"> <PARAM NAME="HTML" VALUE="About Us">
About Us </SERVLET></a>
<a href="Catalogue/"><SERVLET CODE="theme"> <PARAM NAME="HTML"
VALUE="Catalogue"> Catalogue </SERVLET></a>
<a href="comingsoon/"><SERVLET CODE="theme"> <PARAM NAME="HTML" VALUE="Coming
Soon"> Coming Soon </SERVLET></a>
<!-- Note: The text between the <PARAM> and the </SERVLET> tags is not really
needed, as once the server is running correctly, they will never be seen (as
it does not rely on the browser, like applets.  They are present only for
completeness. -->
</BODY></HTML>
```

Figure 2: Example web page for Web themes

Theme	ID	HTML
Text	Body	<body>
Text	About Us	About Us
Text	Catalogue	Catalogue
Text	Coming Soon	Coming Soon
Graphics	Body	<body background="/images/background.gif">
Graphics	About Us	<img src="/images/about_us.gif" width="90" height="20" border="0">
Graphics	Catalogue	<img src="/images/catalogue.gif" width="90" height="20" border="0">
Graphics	Coming Soon	<img src="/images/coming_soon.gif" width="90" height="20" border="0">

Figure 3: Example of the Web theme data

Web themes were looked at as a possible help for web sites that maintain multiple versions of pages, each supporting different options, such as text only pages, and graphical pages This is quite an important thing to do because not all web browsers support all of the latest display options (for example, graphics, sound, non-HTML formats, etc) Effectively web themes split the content of a web site from the context, which means that there is only one copy of the content to keep up to date, as when multiple copies are used, it can become difficult to keep all of them up to date with each other. The web themes engine used SSI (Server Side Includes) in the first prototype, and Java servlets in a later prototype to include the activate content into the web page. This would then fetch the appropriate information from a database and return it to the web browser, thereby allowing one source document to be viewed in several ways, so text and graphical versions of the page, or any other type, could all be created from the same content.

Although the idea was to split the content from the context, if the pages were interpreted instead of including dynamic content, it would be possible to layer both the content and context. Then it would be possible to create templates for all of the pages on a site, and create pages recursively on the fly by allowing inclusion of sub-site templates for different layout of different sub-sites. For example, there may be one template that controls the overall look of the whole web site, which includes content and context from other authors. Each author's content and context could contain versions for each output type they want to support, which can be repeated recursively for larger sites that have many levels between the full web site layout and the individual authors of each page. This would mean that each document may contain content and context from several different authors, all custom built for the person that is currently viewing the document, but there is only one set of content to keep up to date. It also has the side effect of allowing authors to delegate specific portions of a document to others, while maintaining the overall feel of their portion.

It was decided to leave the idea of web themes at the second prototype stage because there were others with similar ideas at the time (circa 1998), and because ideas were looking promising to follow with regard to research on co-operative authoring and collaboration. Today there are options for using XML (http://www.w3.org/XML/) and transformational XSL (http://www.w3.org/Style/XSL/),
or JSP (http://java.sun.com/products/jsp/) with XML or HTML.

Multi-tiered/Multi-homed web sites

There was some investigation into multi-tiered/multi-homed web sites (some of which is incorporated into co-operative authoring). Multi-tiered/multi-homed web sites are those that use redundancy, in this case, redundancy of information to achieve better performance and/or better reliability. A multi-tiered web site splits different tasks (for example displaying the static site and displaying the dynamic site) off to different web servers, which may be on the same machine, or on other machines. This allows the most requested content to be redundantly available, while keeping the advantages of both the static and dynamic sites. Splitting tasks in this manner allows a better load-balancing scheme, which means faster response to user queries, and higher reliability, because if one component fails, it is much less lightly to cause the rest to fail.

A multi-homed server takes this to another level. With a multi-homed server, not only is each task split off to a new web server, but also each web server is in a separate location, possibly with different bandwidth connections. This may not make much sense if the server tasks are mutually dependent - if, for example one is serving the text of a page, and the other the images. If, on the other hand, the servers are performing independent tasks - say one server is set up to allow documents to be read, and the other is set up to allow those documents to be edited - then if one of the servers (or connections to the server), is disrupted, the functions provided by the other server are still available.

A multi-tiered/multi-homed web site would probably be more complex than would be required for some web sites, but for anyone who depends on the reliability of their web site (or web site connection), this could be vital.

After investigating the possibility of doing research along the lines of multi-homed/multi-tiered web sites, it was decided to concentrate instead on co-operative authoring and collaboration, but use some of the knowledge gained from multi-homed/multi-tiered web sites in the co-operative authoring and collaboration research.

Recently a company called Akamai Technologies Inc (http://www.akamai.com/) started to deliver a product that is similar to the system described above, using a massive distributed network of web servers. Akamai's aim is to "deliver a better Internet", by linking you to the nearest copy of the web site, thereby providing faster and more reliable results.

Co-operative authoring and collaboration

Out of the exploration of the origins of the Web, and the exploration of web themes and multi-tier web sites, came the idea to work on the area of web collaboration. The idea of web collaboration sparked some interest because it seemed to be an area that had not had much work done on it since the Web was commercialised. The idea of Web collaboration had some of its roots back in the original ideas for the Web, as well as paper based document creation, and it has the potential to be a very useful contribution to the Internet community.

The ideas about collaboration were coloured by the fact that there are many options for collaboration already available over the Internet. Options for collaboration over the Internet include using IRC to talk (or type) interactively, email and Usenet for less immediate communications, and web pages for demonstrations and the like, these are just a few of the possibilities. These options meant that the ideas for collaboration over the Web had to be refined further than just collaboration. The ideas were refined through to collaboration on particular documents, or in other words, more along the lines of co-operative authoring, rather than the support role that these other methods of communication provide.

BSCW / CSCW

As work was proceeding on YEdit, CSCW (Computer Supported Co-operative Work) and BSCW (Basic Support for Co-operative Work) were discovered. BSCW enables collaboration over the Web and is a 'shared workspace' system that supports storage and retrieval of documents and sharing information within a group. This is integrated with an event notification and group management mechanism in order to provide all users with awareness of others activities in the shared workspace. In particular BSCW is a networked support mechanism, which provides support for co-operative document creation, (for example threaded discussions, group management, search features, and more) and therefore would make a good fit (if the other methods of support are either not appropriate, or alternatives are looked for) with co-operative authoring over the Web.

WebDAV

Another project that was discovered was WebDAV [RFC2518]. Their goal (from their charter) is to "define the HTTP extensions necessary to enable distributed web authoring tools to be broadly interoperable, while supporting user needs". Again, as in the case of BSCW, this project is looking at a different area from collaborative authoring. In particular, the authors are defining a protocol for use by software to connect over the Web. The protocol includes some information that will be helpful for distributed authoring and versioning. It is very possible that in the future, any Web communication could use this protocol to communicate authoring and versioning information. WebDAV is a protocol, more specifically, it is an extension of HTTP/1.1, which adds new HTTP methods and headers to the HTTP protocol, as well as how to format the request headers and bodies. The headers that it current adds are locking (long duration shared/exclusive locking), properties (XML metadata, that is, information about the data), and namespace manipulation (for move and copy operations and for collections, that are similar to file directories). Several extensions are planned, such as advanced collections (which adds support for ordered collections and symbolic links), versioning and configuration management (which adds headers for operations such as history lists, check ins and check outs, both at the individual object level and the collection level), and access control (for controlling access to resources).

The YEdit system that is being created in this research has the capability to be combined with BSCW and CSCW systems to enhance the abilities of both and the potential is there for that, but at this stage there are no plans to combine them. Also YEdit has the potential to use the WebDAV protocol to communicate from clients to the web servers. In the future, when the versioning part of the WebDAV protocol is decided on, YEdit will support it to enable other applications to access YEdit.

The YEdit engine has been created in Java using a combination of JBuilder (versions 2-3.5) and a standard notepad program. It has been tested on a few test systems, three systems running Red Hat Linux (versions 4.x - 6.2) running Apache (versions 1.2 - 1.3) with Apache JServ (versions 0.9 - 1.0), one of which started with the earlier versions of the software and was upgraded, the other two started with the later versions. Two were running Windows 95 for testing. The test systems were used to test ideas and code to check that it worked properly before it was uploaded to the main server at http://www.yedit.com/ which is running FreeBSD 2.2.8-RELEASE, Apache 1.3.9 with Apache JServ 1.0.

4.1.1 Web-based access

The engine that has been created as part of this research is currently primarily set up to work over the Web, but it is not restricted to serving HTML pages. It allows for both the user access and the storage methods to be specified in the initialisation file, along with the name of the class that implements the functionality to either access the storage system, or display the results. Due to the design, this engine is not limited to serving web pages over the Web; it can serve any type of page (be it text, graphics, or other), from any storage system that an interface is created for. At present the engine assumes that pages are HTML text, but that is only a design decision to ease the testing. To

allow any type of page to be stored (graphics, or other), a there are a small number of lines in the source code that need to be replaced with code that determines the content type from the file, rather than assuming that it is HTML.

Web-based access is currently the main method of accessing this engine, allowing pages to be created, edited, and deleted (that is, replaced with a blank page; the old versions are still present). An author can currently create pages with any content, although in the future code that is inserted may be screened for malicious content. The design of the engine is such that pages should be no different from any other general web pages, that is, they should look and act in the same way as any other normal web page. Because of this, web sites that implement this engine should look no different from any other web site that does not implement this engine, at least as far as people who are just browsing. This has important implications in terms of ease of use (there should be no learning curve needed to just browse the pages), and ease of access (anyone or anything, such as web spiders, should have easy non-impeded access to all pages that are available to the general public).

As well as being accessible via any access method (once an interface is created), the engine can also access any type of file storage system. At present the only one supported is a straight file system, but the support is present to allow access to any file storage system, with the creation of some interface code to the system. Probably the next file storage system interface to be created will be one to access a database. This will allow pages to be stored in a database, rather than in the raw file system as is done at the moment. Another useful storage system interface would be one to access a CVS system, which automatically takes care of versioning information, rather than the current system where the servlet records all the versioning information.

4.2 Design of the engine

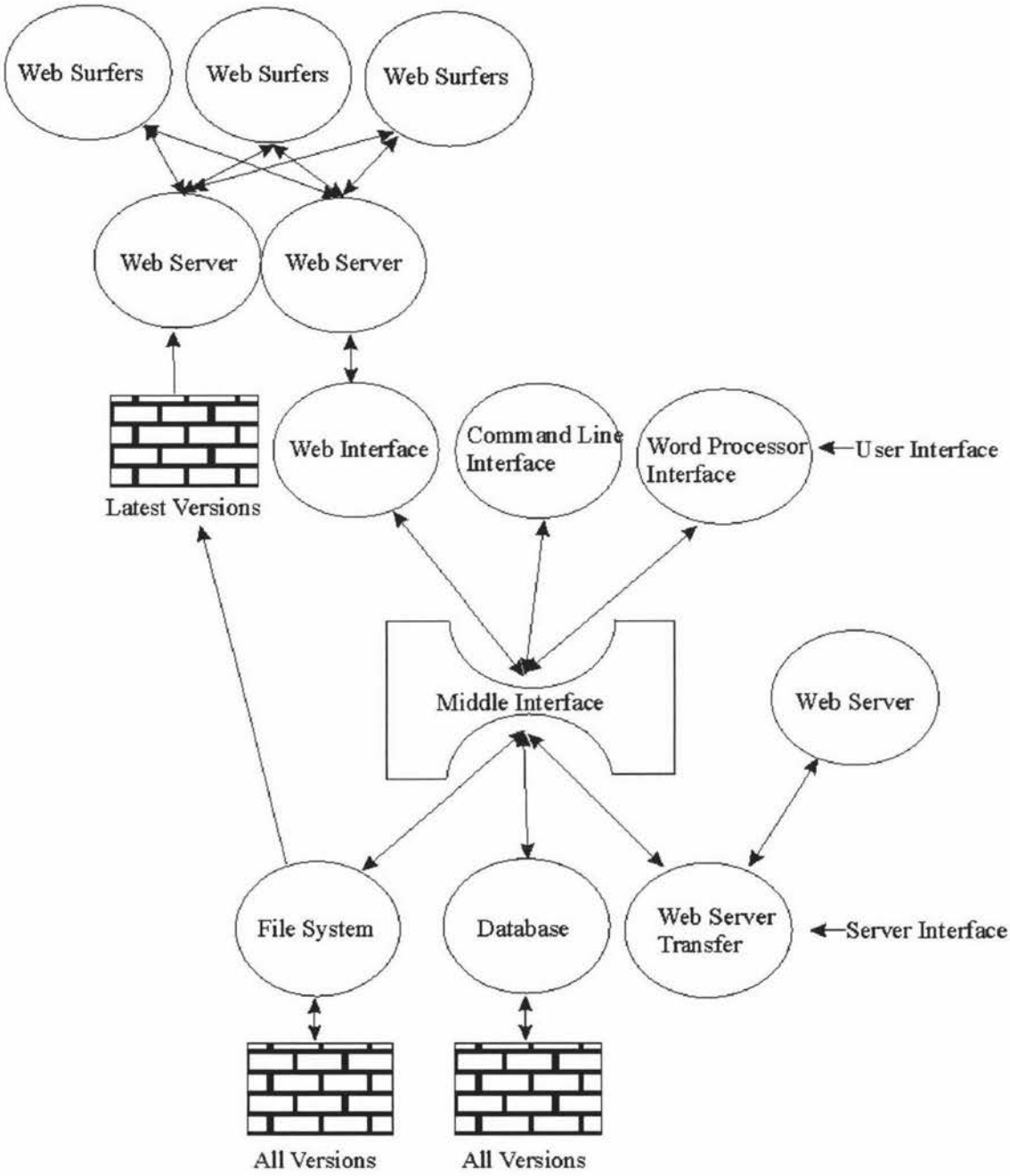


Figure 4: Interaction of the main components of YEdit

The YEdit system is composed of the components in "Figure 4: Interaction of the main components of YEdit". The main components are the "User Interface", the "Middle Interface", and the "Server Interface". Each of these interfaces will be discussed in the following section. As can be seen from the figure, the Web interface is just one of several possible methods of interacting with this system, two other methods of interacting will be using standalone programs, both GUI and command line, and directly with word processors. The same flexibility is contained in the server interface, with the ability to access other storage options such as CVS repositories, or databases.

The User interface

The user interface is the interface that people see and interact with. The ideal user interface is one that does not get in the way of the user using it. The user understands it intuitively, so that the user interface allows work to be done, without intruding on the user's experience of the site.

The main objectives for the user interface are to translate the requests from the user into requests for the "middle interface" (which is located between the user interface and the server interface, and acts as a middleman between them), and to return results from the middle interface to the user.

The Web user interface (as shown in Figure 5: The Web Interface servlets) has three main components. Those components are the read servlet, the browse servlet and the edit servlet, along with a couple of support servlets that are not directly required for using the engine (such as the user preference servlet and the survey servlet).

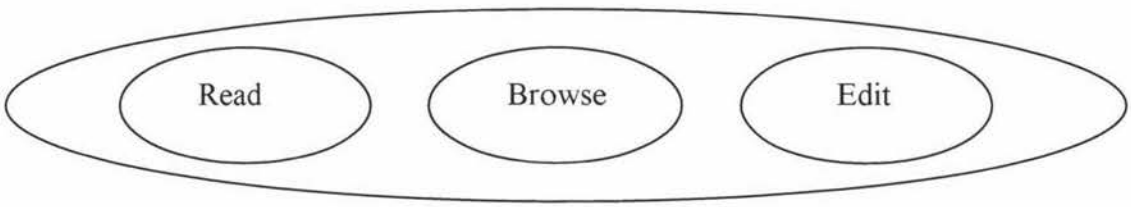


Figure 5: The Web Interface servlets

The Web user interface has been designed to interact over the Web, and works in conjunction with a web server to interface to the user's web browser. This allows access to the engine from any system that supports a web browser. The user interface talks with the web server using the CGI (http://www.w3.org/CGI/) protocol, which defines the interactions between the web server and the web server application. There is a servlet API that abstracts some of the CGI protocol, which makes the interaction easier, but it is still based on the CGI protocol, so knowledge of it is required for the creation of Java Servlets.

In the future, there can be other user interfaces to interact with via means other than the Web. For example there may be direct links into applications, such as word processors, integration into mobile computing, such as Palms and web enabled phones, or direct links to other types of application.

The read servlet is the most transparent of all the servlets to the user. The user accesses web pages through the read servlet, as if they were real static documents available from the web server, with just the name of the servlet before the request. For example the URL to access the web page "/scratch/" (or "/scratch/index.html") would be: http://www.YEdit.com/servlets/Read/scratch/ (or http://www.YEdit.com/servlets/Read/scratch/index.html). This activates the servlet "Read" in the "/servlets" directory of the server "http://www.YEdit.com/".

When the web server receives this request (assuming the web server accesses servlets through the "/servlets" directory, and there is a "Read" servlet present in the servlet directory), it passes the request through the servlet engine, to the servlet (or servlet

alias) that is called "Read". From here, if the servlet has not been started (or if the servlet has changed since it was last called), it is loaded/reloaded and initialised, and then the request is passed on to it. The servlet looks at the parameters that it has been called with (one of which is the path information, that contains the string "/scratch/" or "/scratch/index.html"), and then works out the name of the file that it needs to read in order to satisfy this request. The servlet looks up any path information that needs to be pre-pended to the path information that was passed when the servlet was run, and constructs the full path for the file to be accessed (and adds "index.html" to the end, if the requested path is a directory). The servlet then creates an object that points to this file (whether it is in the file system, a database, or elsewhere). This object has been named for the sake of convenience an USL. An USL (Universal Stream Locator) is a name that has been given to an object that provides information on how to locate a source of information, such as a database, a file, a web page, etc. It is similar to an URL, but can contain any information to access an object, is completely system independent, does not rely on the pointing information being a string, and does not assume any particular method of access. The USL is then passed to the middle interface, which locates and returns the correct class to access the file (currently based on the initialisation file), and then uses that class to access the file and to read the latest version of the file. The read servlet then returns the web page to the user, as if it had been accessed directly. In the future, the read servlet may not be required, as when the edit servlet updates a file, it may also store the file in a directory that is directly accessible by the web server, thereby removing most of the need for a read servlet.

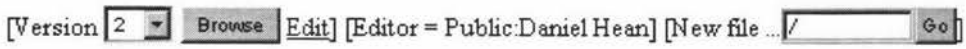


Figure 6: The toolbar that appears at the top of the document in Browse mode

The browse servlet is the one that people interact with when they are looking either for information about the web page (who the author of a particular version was, what a previous version looked like, when it was changed), or are interested in editing a web page. The browse servlet works much like the read servlet does, in terms of accessing a document. The browse servlet adds functions that are not available to users who are just reading the web pages, by adding a toolbar (Figure 6: The toolbar that appears at the top of the document in Browse mode) at the top of the document. The current set of information that this toolbar contains is the current version number of the document and this version's author. There are links to access other versions of the document, to jump straight to other web pages, and if this is the most recent version, then a link to edit this page also appears. If the page does not yet exist, a link is included to add the page.

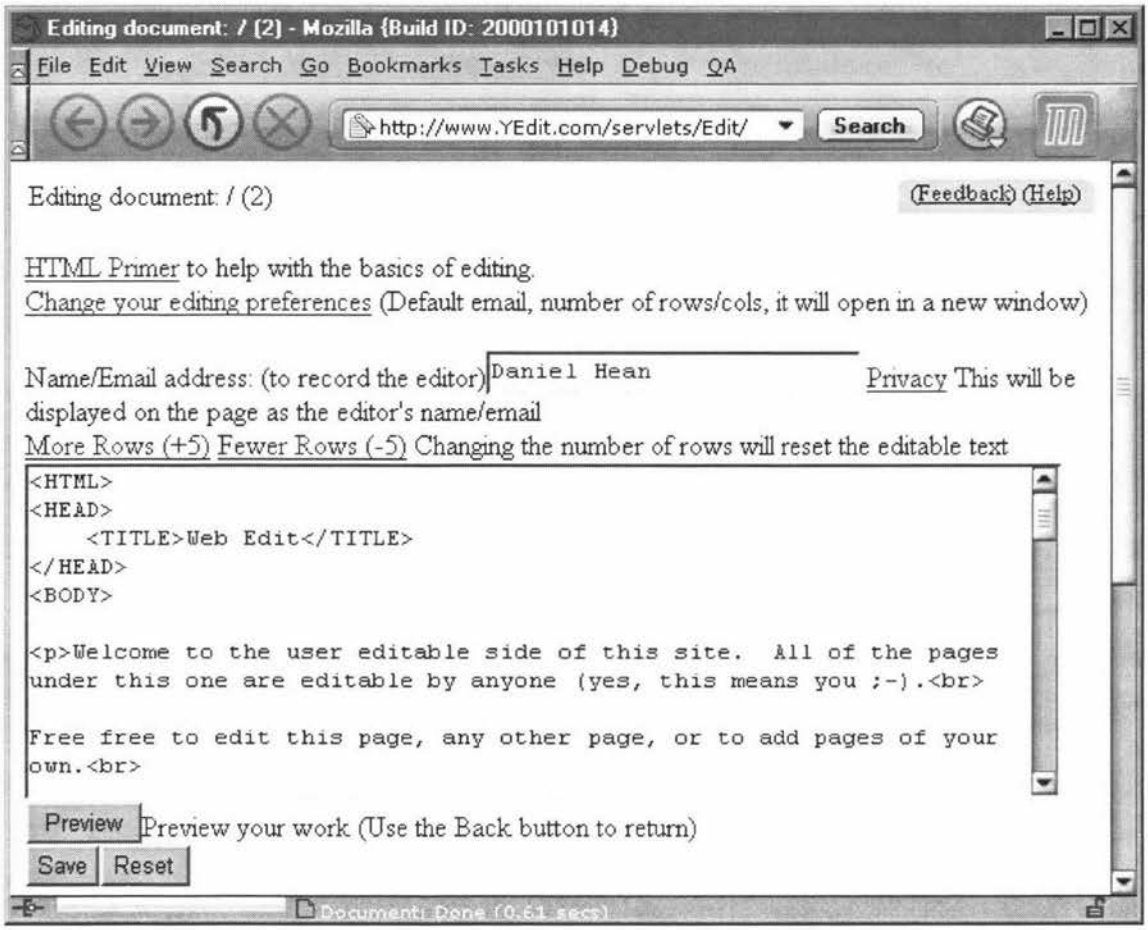


Figure 7: The edit screen

The edit servlet allows people to edit the web pages. It displays a page like the one in Figure 7: The edit screen. At present it returns a web page with various links and text areas for the author's name or email address and the text of the web page. There is also a quick survey at the bottom that requests information about how they find the site. There are links to the user's preferences (email, width and height of text area to edit in), help for HTML editing and links to make the text box for editing larger or smaller.

Users can set a flag by visiting and filling in the user preference web page (that allows people to set up their name or email as their identification, along with the preferred size for the text editing box). This means that when they later edit a web page, the text box containing the HTML will be the size that they set, and their identification will already be filled in (it can be changed at that point if required).

At the moment, anyone editing a page needs to have some knowledge of HTML, as the edit servlet displays HTML in an editable text box. This is a disadvantage to those that do not know HTML, although for some who know a bit about HTML, but not a lot, they could cut and paste between the text box and an HTML editor, but this does not help those that do not know any HTML. This design allows the ability to fully edit every page, while leaving the possibilities for future enhancement by using other methods (for example WebDAV).

After editing the page, the user can use the "Preview" button that is located underneath the text box to preview what the page will look like when saved. If the preview is

satisfactory, they can use the back button to return to the edit page, and can then use the "Save" button to save their work. If their work saves correctly, they will receive a screen that gives them the option to either go back to browsing, or to go back to reading the web pages. If the save did not work, a reason will be displayed. If the save failed because someone else had edited (and saved) the document in the meantime, a page with both users' changes will be displayed. The changes can then be transferred to the current version, and resaved.

When the edit servlet receives a request, it loads the document, just like the read and browse servlets do. Once the document has been loaded, it displays the screen shown in Figure 7: The edit screen, with information to help editing (for example links to HTML help, user preferences, etc), as well as a text box for the editor's name, and a text box that displays the raw HTML of the web page that is being edited.

When the user has finished editing, or just wants to see what the current changes look like, they can press the "Preview" button. When the "Preview" button is pressed, the text that has been entered into the text box is taken and reformatted as a web page and sent back, as the new web page. To go back to editing, the user needs to press the "Back" button on their browser. One very important thing to note, is that the edit servlet **must** ensure that the edit page is cached, otherwise, when the user previews their work, all changes to the page will be lost when they go back to the edit page (using the "Back" button). The only thing the "Preview" button does is to display the changes to the user, it does not save anything on the web server, and therefore it is important that the user does go back and save their work.

Once the user presses the "Save" button, the edit servlet tries to save the changes that the user has made. It uses the class that was retrieved when the document was loaded, and uses it to save the document. In the process of saving, the version numbers are checked, to ensure that the version being saved is still the most recent version. If the version being saved is the most recent version, then it is saved, increasing the version number, and returning to the user a screen saying that the save succeeded, and giving them the option to either continue browsing, or to continue reading. If the save did not succeed, then the user is given an explanation. If the save failed because the document had been updated by someone else in-between the time that the current user started to edit, and saved it, then they are given the option to make the same changes to the new document. This is achieved by displaying a similar screen as to the normal edit screen, but with the changes that they made (that were not saved), as an extra text box on the screen. This allows them to copy the appropriate changes back into the current version, and to then save it.

The Middle interface

The middle interface defines the access that the user interface has to the underlying system. The reason for having a middle interface is to allow the user and server interfaces to be as decoupled as possible, allowing either or both the user and server interfaces to be changed, or replaced. This means that the user interface has no knowledge (and no need for knowledge) of the server interface, where, or how the files are being stored. Also the server interface has no knowledge (and no need for knowledge) of the user interface, where or how the files are being displayed and edited.

This split allows the engine to be used to store any type of document (web page, word processed document, etc), in any kind of storage system (the normal file system, a database, another web site, etc), using any kind of display, or user, interface (web site, command line, word processor). This means that the engine can be used for much more than the storage of web pages, and can in fact be used for all manner of documents that require co-operative authoring and collaboration to work on, which includes versioning. Because of the design of this engine, both user and server interfaces can be written to seamlessly integrate the benefits of co-operative authoring and collaboration into legacy applications that may not fully support co-operative authoring at present, such as web sites.

The middle interface specifies the actions that are available to the user interface to manipulate documents, for example loading and saving, version and author information. While the middle interface provides a selection of methods to the user interface to manipulate documents, it requires certain methods to be present in the server interface to provide the functionality to complete the user interface's requests. Not all methods that the middle interface supports need to be present in the server interface, as some of the methods provided are supplied in the middle interface, using other lower-level methods in the server interface.

The Server interface

The server interface is the interface to the file storage system (whether it is the normal file system, a database, or other versioning repository) that stores the information that is passed through the middle interface. Because the server interface is only accessed through the middle interface, this allows any server storage to be used, and/or interchanged at will, with no visible change to the users, and allows new storage options to be used as and when they become available. Because all access comes through the middle interface, all file storage systems should be compatible with each other.

The server interface has a specific set of methods that it must define and for which it must return the appropriate result, but not all methods that the middle interface supports need to be provided. This is because the middle interface has a default set of actions for specific methods that use the lower-level methods that are supplied by the server interface. There are some methods available in the middle interface that do not need to be implemented if their functionality is not required. There are also some that have default functionality, but should be extended by the server interface to allow the full range of responses, provided that the file storage system can supply the appropriate results.

The rest of the engine

Those interfaces (described in the preceding three sections, the user, middle, and sever interfaces) do the majority of the work in the servlet application. There are other servlets that help these three, and enable other features such as error logging and survey responses.

The Preference servlet looks after all the preferences and options for the system. This stores the information about the server, such as the name of the static and dynamic web

sites, directory locations, and file names. This preference file is first looked for by the servlet in the location that the servlet is being run from, and then the home directory for the user and finally the main Java home directory. The preference file is loaded each time the servlet is loaded or reloaded.

The USL part of the engine is a replacement for URLs, mainly based on the fact that the information sources that can be used by the engine are not limited to normal files and web sites that can be normally accessed be a URL.

The utility classes include the logging and debugging class that makes it easier to locate and fix problems by directing specific debugging information to a specified place.

4.2.1 Java Servlets

The YEdit engine is written in Java utilising the server side technology of Servlets [Servlets] (Java Servlets are Java applications that run on a web server. They are similar to other CGI web server applications). There are many languages that could have been chosen for implementing the engine on the server side, but Java Servlets had the best match between the requirements of YEdit system (which are that the system be relatively simple to extend, can be extended on the fly, efficient for multiple uses of the system at once, and is portable to different operating systems), the programming language, and my knowledge of different programming languages at the time.

Some of the main reasons for choosing Java for the engine were because the programming language:

- is simple (compared with languages such as C++), as it omits some of the less understood and confusing features.
- is object oriented (an object is defined as "an entity that has state; is characterised by the actions that it suffers and that it requires of other objects; is an instance of some class; is denoted by a name; has restricted visibility of and by other objects; may be viewed either by its specification or by its implementation" [Booch 1986]).
- is dynamic, which allows new classes to be created and loaded at run time.
- has networking built in, so connections over the network (Web and otherwise), are much simpler to use than some other languages, no extra code is needed
- is robust. It is much more likely that if there is an error, it will show itself. Add to this garbage collection and memory management, which improve the robustness of the applications and means that errors are less likely to occur in those areas.
- is portable and architecture neutral, which means that once the code is written, it will work on any operating system or machine that supports the Java Virtual Machine (which is potentially most machines, from watches, fridges, phones, etc, as well as conventional computers, whatever operating system they use)

- is multi-threaded, so that more than one request can be run at the same time. This allows much better performance than a single thread, especially when each thread may not be at full working capacity the whole time (for example, when waiting for a file to load or save). This means that more people can use the server at a time

- with respect to servlets, has persistence, that is, once the overhead of loading the servlet is completed, it does not need to be repeated unless one of the classes is auto-reloaded because the class file has been updated

4.3 Current progress

A fully working prototype of the YEdit engine has been created and is available to everyone world wide through http://www.YEdit.com/. This prototype works by implementing one version of both the server and the user interfaces. The user interface is a web-based interface that allows anyone with a web browser to access both the information about this research and the prototype in which they can create and edit web pages. This is achieved using the three user interface servlets, the read, browse and edit servlets, to access and manipulate the documents that are stored in the file storage system. The middle interface has all of the functions that are required for the prototype. The middle interface can load (at run time), any server interface that is specified in the initialisation file. The server interface controls the file storage system that is used. For the prototype, the file storage system that is being used is the normal system file system, of the particular operating system that the engine is being run on (at this stage this has included both Unix and Windows). Because a normal file system does not support the full range of versioning and locks for editing specific versions, this is implemented by the servlet.

The current web site has been designed specifically to house this engine, and it has some information about it and how to use it (for example help on the different ways of accessing the documents, reading, browsing, editing, and some pointers to help for HTML). It houses the prototype and allows anyone to access it to try it out. The feedback that I have received from people who have responded about the web site and the engine has been positive. They seem to be interested in the usefulness of the engine, both in terms of the current use on the web site http://www.yedit.com/, and the future use on other web sites.

The engine has been designed to move as many arbitrary limitations such as types of files or edit applications that are supported as possible from either the middle or the server interfaces, to the design of the user interface that accesses the engine. This allows the creator of the user interface to support any and all types of documents. In principle the engine can handle any type of document with complete transparency. This potentially also includes streaming media (a sequence of video and/or audio that is sent in a compressed form over a network and is decoded and displayed as it arrives, rather than the traditional method of downloading the whole clip, and then viewing it).

The engine has been written in Java because the main considerations of this engine match the strengths of Java. Some of these strengths are the network ability, the robustness and portable nature of Java, and especially servlet persistence in web servers, and the cross-platform write-once, run-anywhere ability. The write-once, run-anywhere ability will by itself ease the maintenance of the engine.

4.4 Testing

Testing [Whittaker 2000] has taken place throughout the whole time that the engine has been in the process of being created. This includes tests to see what could possibly be done, testing individual components of the engine one at a time, thorough to testing each servlet and finally the testing of the whole engine.

Testing of the components involved creating test cases and wrappers for the code to simulate the inputs that would be expected, and to check that the outputs were reasonable. For example, as part of the testing of the file system server interface, the first things that were tested were that the code was receiving the correct information, and decoding it correctly. This involved extracting the path information from the request, extracting the location of the files on disk, the combination of that information, and checking all the way through to make sure that the file system server interface had decoded the information correctly. Once that was tested and working, the next thing to test was the reading of the files, to check that they were being found, and read correctly. Of course just because testing is concentrating on a different part of the code, does not mean that the rest of the code is not being checked at the same time. All that it means is that the rest of the code is not under as much inspection as the code being tested. A lot of the tests were left in the code while other parts of the code were being tested. The reason for this was because if one part breaks, it can be useful to see what the results in the rest of the code are. This is especially useful to double check that other error checking code is being checked for as many of the range of cases as is possible. Another reason for leaving tests in during the testing of other parts of the code is because some errors are complex, and may be within the correct range for each individual part, but when the separately tested modules are all linked together the wrong output is generated. This can often be the case when the information has been manipulated in a technically correct way, but a particular sequence had an unforeseen consequence. By examining the results of the tests in the rest of the code it is often possible to locate the source of the problem, or at least where the erroneous interaction is occurring.

Full testing of the engine has occurred on several of the test machines, and the main server at http://www.yedit.com/. This testing has involved asking people who could be interested in the engine to try it out and see what they think about it.

The first seven test subjects were asked to try out the web based co-operative authoring system, set up inside the Massey firewall, so only those that were located inside the firewall could get access to the machine. Out of those seven people asked to have a look and test the site, five visited the test web site, four filled in a simple survey to gauge their knowledge and experience of computers, and four (including the person who did not fill out a survey form) actually tried out the editing of any of the web pages. From the web server logs, it seems as though they only visited once, and then never went back. This could be because of the pressures that people face at Massey at the present time, or because they do not need the abilities of YEdit at this stage. There were a couple of comments on the forms and in the web pages about the engine. The comments that were left by this group are similar to the comments that I have received about the public site. The comments have been along two main themes the first being that the ideas for co-operative authoring are very good, and the second being that they would like an easier method of editing the web pages, rather than having to know all the HTML for a page. I learnt some valuable lessons (and information) from this testing.

The idea for a simpler method for editing the web pages is one thing that will be followed up after this research is complete.

Further testing is taking place at http://www.YEdit.com/, and some limited promotion has been undertaken to let groups that could be interested in the YEdit system know that it exists. One such group is the people who visit WikiWiki (a user editable, pattern oriented web space), and I have established links from there that mention the YEdit system and provide links to http://www.YEdit.com/.

Testing at the public web site has progressed slowly, but has produced some useful results, such as discovering an unintended interaction between the last date of modification and caches between the engine and the user. This interaction never seemed to produce an error on the server side, but produced an "all Ok" error on the client side. This is a **very** peculiar error to get, because "all Ok" means the request succeeded and the result is provided, but in this case the "all Ok" status was somehow getting interpreted and displayed as an error, which should never happen, as it is not an error. This has since been rectified by not supplying the last modified date as part of the headers sent back with the document. The use of http://www.YEdit.com/ is slowly increasing as efforts to promote the web site are progressing. A discussion of web site promotion is beyond the topic of this thesis.

4.4.1 Creating a testbed

Testbeds were used in the creation of the YEdit system for testing changes and new ideas. The testbed web servers were fully functional web servers that were placed on local machines to test the responses of the YEdit system. The reason for using local testbed web servers was to test ideas and changes in a fully working environment before they were run live on the public web server.

Some of the reasons for creating a testbed for co-operative authoring and collaboration are to test out ideas and changes, to get a working model, to test the feasibility of concepts and to see what other people think of the ideas. Also it allows a small number of people to test and refine new ideas before they are moved to the public web server.

The testbeds were created so that YEdit system could be plugged into a web server, and it could be tested and re-written as the process continued around the iterative spiral of ideas, design through testing. Because of the development path suggested by the spiral model, the ideas, design, coding, testing, etc are all spread throughout the whole project. This means that it is important to have testbeds where the changes and new ideas can be tested and refined before they are finalised for the public web server. Effectively this means that the project has been through many phases of idea generation, design, coding and testing. Because of this various different approaches have been tried for many different parts of the project. For example, some of the early prototypes had options and directory structures hard coded into the code (as they were only there for testing ideas), and as the ideas worked, the options and directory structures were moved out to a initialisation file, in progressive steps. In the current version all the directory structures are stored in the initialisation file, along with many other options, such as method of access to the version storage area, and the classes used to access it.

The initialisation file has been further extended to generalise the features. The current version supports accessing various initialisation files, for example there may be one per person, one per application, along with ones for the system as a whole, and custom settings based on both the person and the application. This generalisation will allow the initialisation files to be used in a more general context, as well as supporting both web and non-web access to the versioning storage area.

As an example, here is a portion of the initialisation file that is specific to co-operative authoring. This is the portion that defines where the file storage, that stores the versions, is located. As can be seen in the file (Figure 8: Example portion of an initialisation file), it can store information for more than one system, which allows the same initialisation file to be used on different systems with ease. The information about the storage location is supplied with the "`Collaboration.storage`" option; this allows the choice of system that YEdit is currently running on. The options for this particular file are "`File.YEdit`" for running on the public web server or "`File.localhost`" for running on one of the local test web servers. As can be seen by the options below that, the option for "`Collaboration.storage`" is then used to select the correct file locations and code to access that location. A full example of an initialisation file is located in the appendix - 7.1 Appendix A: A full example of a preference file.

```
##---Program wide Preferences (properties)---
##Location=Jar:/Preference/

##Default systems for the storage location
Collaboration.storage=File.YEdit

##File system storage
Collaboration.File.localhost.BaseServer=D
Collaboration.File.localhost.BaseDir=\\Software\\Java\\revisions\\
Collaboration.File.YEdit.BaseDir=/usr/home/yedit/revisions/
FactoryAccess.File.localhost=com.YEdit.ServerInterface.FileSystem.FileSystem
FactoryAccess.File.YEdit=com.YEdit.ServerInterface.FileSystem.FileSystem
```

Figure 8: Example portion of an initialisation file

The project is accessible worldwide (at http://www.YEdit.com/). The local testbeds are now used to test changes and ideas before they are put onto the public server. The local testbeds are especially useful for tests that would not be advisable to run over the Internet. These are tests like loading response (testing what happens under heavy load), and any other tests that may interfere with either the other hosts on the same machine as YEdit, or the machines in-between the two systems being tested.

4.5 Summary

The design model used for the creation of the co-operative authoring and collaboration engine was the spiral design model because the iterative model of creation and orderly refactorings, were a good match for the project.

Some other areas were looked at in the process of refining the initial idea of enhancing communication over the Web. One of the first was the presentation of information, the split between the information (content) and the display environment (context). Test prototypes were created that used this split to enable one version of the content to be used for both the text only and the graphical displays.

Another area that was looked at was multi-tiered/multi-homed web sites. This was to separate the production of information from the display of that information. This means that one web server could look after all the active content (such as returning all the information from a storage system), and another web server (either on the same machine, or preferable in a different location, with a different Internet connection) could display a copy of the active content, which has been stored in static files. This gives higher reliability, as the area that is most likely to have a problem is the active content (as there is more to go wrong), while the static web site can still display all the information up to the point of the problem. Or if the static site has a problem, all traffic can be moved to the active site.

Two areas that may look similar on the surface were looked at, BSCW/CSCW, and WebDAV which will be discussed in more detail in the following two paragraphs.

BSCW/CSCW (BSCW being the main version of CSCW for the Web) enables collaboration over the Web and is a 'shared workspace' system that supports storage and retrieval of documents and sharing information within a group. This is integrated with an event notification and group management mechanism in order to provide all users with awareness of others activities in the shared workspace. In particular BSCW is a networked support mechanism, which provides support for co-operative document creation, (for example threaded discussions, group management, search features, and more).

WebDAV is a protocol that "defines the HTTP extensions necessary to enable distributed web authoring tools to be broadly interoperable, while supporting user needs". The protocol includes some information that will be helpful for distributed authoring and versioning. It is very possible that in the future, any Web communication could use this protocol to communicate authoring and versioning information. WebDAV is a protocol, more specifically, it is an extension of HTTP/1.1, which adds new HTTP methods, and headers to the HTTP protocol, as well as how to format the request headers and bodies.

The design of the engine was such that the access to the engine, the access to the file storage system, and the interface between them were all separated out. This meant that with the middle interface linking the user and server interfaces, the user and server interfaces could be created separately, and could be changed to suit the environment that surrounds the engine. In order to create this prototype, the server interface was implemented using access to the system's file system, and the user interface was implemented using a web-based interface. The web-based interface has three main

parts (Refer to Figure 4: Interaction of the main components of YEdit). The read servlet displays web pages; the browse servlet allows interaction with the information about the different versions, and access to the edit servlet. The edit servlet allows the web pages to be created and edited. The engine was created using Java servlets, as they provide a good match to the requirements of the engine.

The engine is currently working at http://www.YEdit.com/ and has had user testing both there and on test machines, as well as testing throughout the creation process.

5 Conclusion

Co-operative authoring and collaboration over the World Wide Web using an engine called "YEdit", based at the web site http://www.YEdit.com/, demonstrates the co-operative authoring of individual documents, and collaboration in general. The version that appears there is the latest version that has been tested successfully on the testbed servers. The testbed servers are used to test the code before it is placed on the public server, they are also used to run tests that are better run on a standalone network, because they may interfere with other computers if run over the Internet. Tests like load and stress testing are done in this manner.

YEdit (the prototype system created for this thesis) demonstrates the ability for people to edit documents over the Web, from any location, at any time. It stores documents in the local file system of the computer that is hosting the engine, and serves pages over the Web, with the ability for anyone to fully edit them while keeping full version information. YEdit also allows for documents to be stored in the web server's document tree. This means that the web server can serve the documents as if they were local static documents. This allows the reading of the documents without regard to the status of the YEdit engine, this gives much better reliability than serving the pages dynamically every time.

The YEdit engine allows the free mixing of ideas by allowing anyone to edit any page. People who would like to add their ideas only need to browse the web site, and start editing (whether creating new pages or editing current pages), to add their ideas, or to enhance ideas that are already present. Of course should a Webmaster want to restrict a particular area to read only, or restrict access to only a select few, then the normal access provisions of the web server can be used. This means that YEdit can be used for public editable content, public read-only content, and private content, with the use of the normal access controls that the web server provides.

All of this builds up to a vision of the future for the Web that allows the freedom for groups to work together to not only provide information over the Web, but to provide a means of creating new information at the same time. That is, to allow the flow of information not only from the web server to the web surfer, but between groups of web surfers and web servers. The ability for the information to flow in all directions is enabled by YEdit because it allows people to co-operatively author, and collaborate on, documents that they wish to share the creation of. They can receive comments and additions, with the comments and additions made as part of the actual document, rather than making comments and additions using some other communication channel, like comment forms or email.

One of the early possibilities that was looked into for this thesis was to use web themes on the web server to allow a split between content and context information. This split allows for there to be multiple ways of viewing information, all with one content base, that is, only one file to update that contains the content for the page. The context information (images, buttons, other graphics, etc) could be linked to, and the appropriate version appears when someone views the web page. For example, if the user is browsing in text only mode, all the graphics will be replaced with their text

equivalent, or if it is near a particular time of year, the default images and graphics could be changed to match that time of year. If the user prefers a particular theme or colour scheme, providing the web site supports that theme, the user can view the site using it. Because this is all done on the server side, there are no requirements placed on the user, therefore anyone can use it. Prototypes were created, first in 'C', and then in Java to test the idea and to see if it would work. These themes allow the user to choose the method of display that is the most appropriate and convenient for them, with only one web page for the Webmaster to keep up to date. Thus users can view the page as they wish, and the Webmaster only has to keep one set of information up to date. These prototypes showed that it could work with fuller integration into the web server (rather than calling via SSI's (Server Side Includes; these let a user embed a number of special 'commands' into HTML. When the server reads a SSI document, it looks for these commands and performs the necessary action. For example, there is a SSI command that inserts the document's last modification time. When the server reads a file with this command in, it replaces the command with the appropriate time.) or Servlets as was done in the prototypes). This idea was considered for use in this thesis, but at the time (circa 1998) others were floating ideas of using templates and similar ideas for a similar job. Since then other technologies, such as XSL/XML, JSP/XML, or JSP/HTML, for separating content from context, have maturated. Unfortunately it is impossible to tell if these technologies are widely used (or used at all), because the entire interaction happens on the web server, before the user sees the page.

Along with web themes, Multi-tier and multi-homed web sites were looked at early on. The thinking behind multi-tiered and/or multi-homed web sites is that in case one part of the site fails or is lost in some manner, another part can take over at least some of the responsibilities of the failed part. That is to display and/or store information so that it is both stored safely and accessible. This had potential to be used by itself, or in combination with other ideas. Using multi-tiered/multi-homed web sites in combination with other ideas such as the free flowing web of information allows for a more robust system. This is because it tolerates any problems with the dynamic web site (that may be encountered in any newly designed system that are results of unexpected situations), as most people will be viewing the web pages from the static read-only web site, while editing is done on the dynamic site. This allows both the read and edit parts of the dynamic web site to crash, while only inconveniencing a small proportion of users. The level of service and information that is replicated over the different locations determines the level of redundancy offered by a multi-tier/multi-homed web site. The ability to access the latest version of a web page is the most important one to have available, as most people are probably surfing the Web, not expecting to be able to edit it. This is backed up by the access patterns on http://www.YEdit.com/. In the past 2.5 months, people have viewed 2729 web pages on the web site, while only making 25 page edits during that time, which is approximately 1% of page views were to edit a web page. This page view and edit information shows that if there are any problems, the way to minimise the impact is to make sure that the majority of people can continue to do what they were doing, and that is to surf the Web (in this case, view the read-only pages). This means that the most used part of the YEdit engine, and therefore the part that is most important to replicate and keep redundantly available is the ability to view the latest version of the web pages. This split between reading the latest version of a web page, and browsing all the versions and editing them, is used in the YEdit engine to allow a multi-tier, or multi-homed system to exist, and allows for the redundant storage and viewing of the latest versions. Another reason for the split and redundant

availability of the web pages is so that any problems in the more complicated subsystem detract as little as possible from the overall system. There are other multi-tier/multi-homed systems that can be used (and could be used in conjunction with YEdit), such as load balancing between machines, mirror web sites, and lately Akamai Technologies (who have web servers located globally, and replicate web sites to the servers that are closest to users).

The YEdit engine was created in Java, using Java Servlets. At the time, Java was a very new language, but it had good support and several advantages over other languages such as Perl or C/C++ for server side applications. There are some very important considerations when writing applications that are to run on a web server. Some of these are robustness, abilities of the language, portably, support, and general susceptibility to errors, especially buffer overruns, as they are one of the most common exploits. One of the important things for YEdit is the dynamic object nature of Java. C++ has similar abilities, but lacks the easy portability, and can have problems with security (especially buffer overruns). Perl seemed to lack some of the abilities (such as reducing heavy load on the web server, unless the server supports persistence for Perl, like "mod_perl" in Apache, and less opportunity for dynamic object oriented code loaded on the fly), while Java Servlets stood out with a very good match for the needed abilities, with the advantage of a similar syntax to C/C++, which eased the initial learning of Java. As Java was such a young language at the time, it was important that the reasons for choosing it were because it did what was needed, as has been the case. Today it is a much more accepted language, and over the past years, the support for Java Servlets has increased in the commercial web server market. There are now a reasonable number of web hosts to choose from, whereas when this was started, there were hardly even a handful to choose from, and if you wanted Java Servlet support, it was almost a case of setting it up yourself, which actually is not all that hard.

The YEdit engine is flexible enough to be used for a wide range of different document types, from web pages, to word processor documents, to any kind of document that one wishes to edit co-operatively. It also has the capability to store the documents in any kind of repository that an interface can be created for. This potentially allows it to access and store information in any current document storage system, and to serve the stored documents from this system to any document editing facilities that are available. This allows any group or business to have the full advantages of co-operative authoring, while keeping prior investments in legacy technology intact. It enhances their ability to advance the technology they use to the latest that is available, without many of the normal side effects of changing the software that people are used to using. The side effects are mitigated because most users can continue to use the software that they are familiar with, while allowing the software that is working in the background to be advanced and upgraded. Because the user interface and server interfaces of the YEdit engine are independent, there is no reason that a particular storage system should be used for any particular software that a user is using, this allows for the storage system to be completely upgraded with little impact on the users. This de-coupling of the storage system from the user-editing program also has the side effect of allowing an user interface for the YEdit engine to be created that can automatically convert documents to a standard format for storage. This can have the benefit of allowing user-editing programs that store files in different formats to be used, accessing the documents transparently to the user. The document may have been edited in a completely different

editing program, as YEdit could provide the document in a particular programs native format, without the need for the user to intervene.

YEdit is flexible enough to support future expansion and development in methods for storing and serving documents. It was designed to work over the Web to begin with, to test and evaluate possibilities for use and to allow people to become familiar with this type of system. The Web is also expected to be the most used interface, therefore it should be the first one created. Included in the future developments for storing documents is using databases to store the documents, and the possibility of storing documents into source code revision storage facilities, such as CVS for advanced version control. Included in future developments for serving documents are a simplified web edit ability (for those that would rather have a simpler way to edit documents), serving and editing documents using WebDAV (an open distributed authoring and versioning protocol), WAP (browsing simplified web pages on mobile devices), and directly into your favourite document editing program, using their native formats.

The major goal of this research was to see if it was possible, and if possible to create, an system that would handle co-operative authoring over the Web. As a secondary goal the system should be flexible enough to potentially handle any kind of document, any kind of storage system, and any kind of user application to access it. Successful implementation of the YEdit engine has demonstrated the viability of those ideas. The secondary goal was also achieved, allowing the flexibility of any kind of document to be stored in any kind of storage system that has an interface created for it, and accessed by any kind of user program that an interface is created for. Now that YEdit has been created, and has interfaces for editing the Web, it will be appropriate to create interfaces to other applications. This is so that YEdit can be used by a variety of different applications, and interfaces to a variety of different storage systems so that when people first hear about YEdit, they know that it can interface with their current set up seamlessly. This will allow them to use the software that they want to, rather than being locked into any legacy system.

Further research could be possible into how a system such as YEdit will affect the use of authoring software in groups, especially from the point of view of businesses, as YEdit has implications on the use of both legacy and current software. Other research areas could include how people use the YEdit interfaces (where they are visible to the end user), and whether the interfaces should be completely transparent, or should give some sign to the user that they are being used. Also there could be research into the refinement of the Web interface, and web page editing interfaces in general for editing documents live on the Web. This would have implications not only for YEdit, but also for web editing in general, and may have a flow on effect to other editing interfaces.

There are other things that have been created in the process of conducting this research that are not directly related to its aim, such as the design and construction of the web pages to support the YEdit engine. These web pages inform people about the uses and abilities of YEdit and these web pages can be enhanced in the future. Along with enhancing the web pages, further promotion of http://www.YEdit.com/ can proceed especially with regard to users who may wish to use the abilities of the YEdit engine to enhance their co-operative authoring and collaboration.

The YEdit engine has succeeded in demonstrating that co-operative authoring of individual documents over the Web and collaboration in general is possible, and that it is possible to create an engine that is general enough that it can support a wide variety of document types. The front end can interface with many different applications, allowing people to work in the manner that best suits them, and the storage interface can store documents in any locations that are appropriate for the documents concerned. The current main user interface is through the Web. This means that it can be widely used and tested without the need to install specific software on users' computers, allowing a broad range of people to try it out and decide if it will be useful for them. The YEdit engine also allows a wide range of document storage systems to be used. The combined abilities to handle any type of document interface with any kind of document storage system, and interface with the users most used applications will allow the YEdit engine to enhance even legacy systems with the maximum of ease and usefulness.

6 Glossary

Brochure site

A web site that only has a brochure for a company. In other words, it has no product or service information, requires you to snail-mail, or ring to get any information, and some may even ask you to email the site to get the information that you are looking for. This is the very information that should be up on the web site in the first place.

CERN

Conseil Européen pour la Recherche Nucléaire (French), European Organisation for Nuclear Research, now known as the European Laboratory for Particle Physics.

See http://www.cern.ch/

CGI

CGI is a protocol that defines the communication between web servers and external executable code on the server.

Client/server

Client/server is a computational architecture that involves client processes requesting service from server processes.

See http://www.abs.net/~lloyd/csfaq.txt

Collaboration

To work jointly with others or together especially in an intellectual endeavour.

Content

Useful information that has substance.

Co-operative authoring

Marked by a willingness and ability to work with others to be a writer (or creator) of a literary work.

Cross platform

The software can run on multiple operating systems, and/or different base hardware.

EdLine

EdLine is a very early text file editor that shipped with early versions of MS-DOS.

Email and Usenet (Newsgroups)

Methods of communication for discussion based on text. Email is for private one on one discussion, and can be used for groups, when a mailing list is set up. Mailing lists are still private to those involved, unless it is publicly mirrored somewhere. Usenet is for open public discussions that everyone can read and respond to. Like verbal discussions, these are not implicitly recorded and kept, unless someone makes the effort to do so (for example Dejanews).

Eye-candy

Something that is only present to please the eye, and has no information content. When done right, this can enhance a site. When done wrong (as is common), it can replace, or destroy any previously present content.

FAQ

Frequently Asked Questions are most often associated with Usenet, although they have grown to cover any area or subject, not just Usenet conduct and frequently asked questions.

See news://news.answers/

See ftp://rtfm.mit.edu/pub/usenet-by-hierarchy/

See http://www.faqs.org/

Fluff

The opposite of 'Content'. Stuff that is there, and gives you no information.

HotJava

A web browser created by Sun that is written entirely in Java.

See http://java.sun.com/

HTML

HyperText Markup Language. The language for publishing web pages on the Web. It is a non-proprietary format based on SGML that can be created by any text editor, through to WYSIWYG publishing tools.

See http://www.w3.org/MarkUp/MarkUp.html

Hypermedia

Hypermedia is a term describing the combination of Hypertext, sound, graphics, etc. It is Hypertext that is not limited to text.

Hypertext

Hypertext is computer-readable text that allows very extensive cross-referencing.

The Internet

The main interconnection of networks of computers (c.f. internet, an interconnection of networks, any kind, any where, and not necessarily global, or connected to the Internet). The Internet grew from the original ARPANET network in the USA (1970s).

Java

Java is a recent programming language that is ideal for network programming, when security and portability are required.

See http://java.sun.com/

Java Servlets

Java Servlets are Java applications that run on a web server. They are similar to other CGI web server applications

See http://java.sun.com/products/servlet/whitepaper.html

Legacy technology

Legacy technology is older technology that has been superseded by newer technology. Commonly associated with technology that may be older, but that still contains needed and useful information, that either can't be upgraded, would cost too much, or there is no need (although there may be wants) to upgrade.

Mosaic

The web browser that started the graphical web trend. This is the browser that both Netscape and Microsoft have based their browsers on, as it was the first and easiest (at the time) in-line graphical browser.

Newbie

Someone who is new to the scene (commonly associated with Usenet). This is one stage that we all go though. This is when understanding about a new subject or group is being discovered. In contrast is the "Clueless Newbie". This is someone who is new, but can't be bothered to find out what is/is not acceptable in the area that they are discovering. In other words someone who is new to the area, and wants everything handed to them on a silver platter. Often associated with Usenet when someone asks a question that is answered in the FAQ, that they should have read.

Open Source

A software development model that harnesses the many programmers spread throughout the Internet, rather than the traditional model of harnessing a few programmers in a specific location.

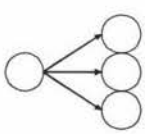
See http://www.opensource.org/

Open standards

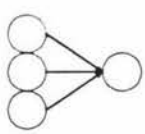
Making a standard available to anyone, this means that there will be much higher support, and the possibility of systems that can easily interact.

Relation ship diagrams

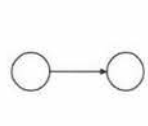
One to Many



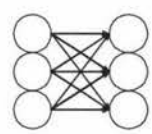
Many to One



One to one



Many to Many



Request For Comments (RFCs) / Internet Standards (STDs)

RFCs are officially available from either the IETF or http://www.rfc-editor.org/, which also lists other repositories.

WebDAV [RFC2518]

HTTP [RFC 2616]

HTML [RFC2854]

Telnet [RFC854-855 / STD8]

FTP [RFC959 / STD9]

Email (SMTP) [RFC821 / STD10]

(POP3) [RFC1939 / STD53]

IRC [RFC2810-3]

Usenet (NNTP) [RFC977/RFC1036]

Gopher [RFC1436]

WAIS [RFC1625]

SGML

Standard Generalised Markup Language

SPAM (& the Real time blacklist)

Unwanted and unasked for email that companies or individuals send out to LARGE lists of people without their consent. This is often in the vain attempt to sell people something that they can't sell (because no one wants it, it's illegal, immoral, or too expensive). The name SPAM comes from a Monty Python script that more than just overuses the word SPAM. There are groups that keep a list of spammer's addresses, called 'Real time blacklists', and many ISP's (Internet Service Providers), will automatically block any emails coming from the known spamming addresses.

See http://spam.abuse.net/

See http://maps.vix.com/

SSI (Server Side Includes)

This allows active content in an otherwise static web page

W3 (World Wide Web Consortium)

The co-ordinating body for Web development.

See http://www.w3.org/

The Web (WWW, World Wide Web)

Collection of servers that serve web pages over the Internet compared with the Internet, which is the collection of networks that the Web works over. The Web is effectively an application running over the Internet.

Web browser

An application that allows you to view web pages.

Web hits

The definition of Web hits can vary, depending on what exactly you are talking about. The term is a very fuzzy one, and very difficult to interpret, even if you are talking about the same meaning. One meaning is one hit is one access to the web site, so for one web page, there might be anything from zero (yes that is right, zero) to twenty or more. Another meaning is one hit per page viewed, with all the graphics and associated pages (for example frames) counted all as one hit. Of course this is all assuming that the page was actually asked for from the web server, and it was not cached in the local cache, the proxy cache, or elsewhere (which is where the zero from above can come from). Then there is the question on whether the machine asking for a page is being read by a person, or is it just a robot going out in search of pages, generally to index them. Then there are the mistaken hits, when someone types in the wrong address, someone else had the address before you, or someone misspelled a link in a page.

Webmaster

The person who is in charge of the web server and looks after all the information and pages that are on it.

Web pages

Pages of information that people can put onto web sites so that others can access that information easily from anywhere, usually written in HTML.

Web site

A computer that is set up to serve web pages.

WikiWiki

Quote from their front page: "(It is) ... a fun way of communicating asynchronously across the network", or as a set of web pages that are open and free for anyone to edit as they wish. Wiki is not real-time; therefore people have time to think before they follow up a web page, often days or weeks, so they have time to consider what they write. It was created for the discussion of People, Projects, and Patterns (a pattern is a recurring solution to a common problem in a given context and system of forces [Alexander 1977][Alexander 1979]).

Word of mouth

Information that people have thought worthy enough to pass on to others that may be interested.

WYSIWYG

What You See Is What You Get. Unfortunately this is not always the case.

7 Appendix

7.1 Appendix A: A full example of a preference file

```
#---Program wide Preferences (properties)---
#Location=Jar:/Preference/

CollaborationJarPreferences=Loaded

## Default systems for the storage and display locations
Collaboration.storage=File.YEdit
#Collaboration.storage=File.Server
#Collaboration.storage=File.localhost

Collaboration.export=Export.YEdit
Collaboration.display=Console.null

## File system storage
Collaboration.File.YEdit.BaseDir=/usr/home/yedit/revisions/
Collaboration.File.Server.BaseDir=/home/drhean/yedit/revisions/
Collaboration.File.localhost.BaseServer=D
Collaboration.File.localhost.BaseDir=\\Software\\Java\\revisions\\
#
FactoryAccess.File.YEdit=com.YEdit.ServerInterface.FileSystem.FileSystem
FactoryAccess.File.Server=com.YEdit.ServerInterface.FileSystem.FileSystem
FactoryAccess.File.localhost=com.YEdit.ServerInterface.FileSystem.FileSystem

## Static web site location
Collaboration.webStatic=www.YEdit.com
#Collaboration.webStatic=Server
#Collaboration.webStatic=localhost

## Dynamic web site location
Collaboration.webDynamic=www.YEdit.com
#Collaboration.webDynamic=Server
#Collaboration.webDynamic=localhost

## Email
Collaboration.email=...@YEdit.com
#Collaboration.email=...@massey.ac.nz

## Servlet dir location
Collaboration.servlets=/servlets
```

```
## Location of the Server Home directory
Collaboration.ServerDir=/usr/home/yedit/
#Collaboration.ServerDir=/home/drhean/yedit/
#Collaboration.ServerDir=D:/SOFTWARE/JAVA/

##File system for Revisions (from the Server dir)
Collaboration.revisionDir=revisions/

## Location to save all the survey inforamtion (from the Server dir)
Collaboration.surveySaveDir=surveys/

## Error/Debug log file (from the Server dir)
Collaboration.log=surveys/log.log

## Location of the WWW Home directory
Collaboration.wwwHomeDir=/usr/home/yedit/www/
#Collaboration.wwwHomeDir=/home/drhean/yedit/www/
#Collaboration.wwwHomeDir=D:/SOFTWARE/JAVA/htdoc.YEdit/

## Filename of the 404 page (from the WWW Home dir)
Collaboration.page404=missing.html

## Filename of the New (Add) page (from the WWW Home dir)
Collaboration.newPage=new.html

## Directory to store the newest export version
Collaboration.Export.YEdit.BaseDir=/usr/home/yedit/www/public/
Collaboration.Export.Server.BaseDir=/home/drhean/yedit/www/public/
Collaboration.Export.localhost.BaseServer=D
Collaboration.Export.localhost.BaseDir=/SOFTWARE/JAVA/htdoc.YEdit/public/
#
FactoryAccess.Export.YEdit=com.YEdit.ServerInterface.FileSystem.FileSystem
FactoryAccess.Export.Server=com.YEdit.ServerInterface.FileSystem.FileSystem
FactoryAccess.Export.localhost=com.YEdit.ServerInterface.FileSystem.FileSystem
```

7.2 Appendix B: Using YEdit

Reading a file using YEdit

The following extract from the "read" servlet demonstrates reading the latest version of a file, and returning it as a string that is then sent to the web browser. The parameter passed as "USL file" is the location of the file that is to be read.

"Web.storageSystem" is the type of file system that is in use. This is read from the preference (initialisation) file.

```
public static String read(USL file)
  {
  com.YEdit.Interface.IAccess storageObj  = null;
  BufferedInputStream bufIn = null;
  int c = 0;
  StringBuffer output = new StringBuffer();
  PrintWriter out = null;
```

Up to here is just the setting up of the variables. The following line creates an object using the correct class (as defined in the preference file) to access the file storage system that is in use.

```
  storageObj = com.YEdit.Interface.FactoryCreate.access(Web.storageSystem,
file);
```

The following "if" checks to see if a specific version is wanted, if not, it retrieves the latest version.

```
  if( null == file.getVersion() )
      {   // Get the latest version
      storageObj = com.YEdit.Interface.FactoryCreate.access(Web.storageSystem,
file.dupVersion(storageObj.highestRevision()));
      }
```

Now that the correct version has been located, it is read in. If there is an error, the error is logged and a blank file is returned.

```
  try {
      bufIn = new java.io.BufferedInputStream(storageObj.getInputStream());
      while ((c = bufIn.read()) != -1)
          {
          output.append((char)c);
          }
      bufIn.close();
      }
  catch (IOException e)
      {// File not found
      Log.error(e, "File not found: " + storageObj.getUSL() );
      out = null;
      return(null);
      }
```

The file is returned as a string, ready to be passed to the web browser.

```
  bufIn = null;
  return(output.toString());
  }
```

Writing a file using YEdit

The following is extracted from the Edit servlet, from where it saves the edited file.

```
  {
  USL outUSL = null;
  String fileText = null;
  IVersion oldFileVersion = null;
  com.YEdit.Interface.IAccess storageObj  = null;
  String user = null;
  String email = null;
```

The above code sets up some of the variables that are used.

"outUSL" is the location of the file to be saved.
"fileText" is the text of the file to be saved.
"oldFileVersion" is the version of the file when it was read (0 if it is a new file).
"email" is the user's name or email address

The following code opens the file to be saved, checks that the version that has been edited is the same as the current version (so as not to overwrite any other changes), and then saves it.

```
  storageObj = com.YEdit.Interface.FactoryCreate.access(Web.storageSystem,
outUSL);
  if( storageObj.highestRevision().equals(oldFileVersion) )
      {   // The edited document version matches the most recent document
      user = request.getRemoteUser();
      if( null == user )
          { user = "Public"; }
      if( null != email )
          { user = user + ": " + email; }
      storageObj = com.YEdit.Interface.FactoryCreate.access(Web.storageSystem,
outUSL.dupVersion(oldFileVersion.next()));
      if( storageObj.lock(user, "Web") )
          {
          try {
              storageObj.print(fileText);
              storageObj.unlock();

              // File saved
```

The file has been successfully saved, therefore success is reported to the user at this point, and control is returned.

```
              return;
              }
```

```
            catch(IOException e)
                {
                // Error writing to the file
```

An error occurred when writing to the file; it may not have been saved. A warning is displayed to the user at this point, and control is returned.

```
                return;
                }
            }
        else{
            // Error locking the file
```

The file could not be locked; maybe it is in use by someone else? A warning is displayed to the user at this point, and control is returned.

```
            return;
            }
        }
    else{// The edited document and most recent document do NOT match
         // This means that someone else has edited and saved the document
```

The document has been updated by someone else, so a message asking the user to please merge both sets of changes into one and save the results is displayed.

```
        }
    }
```

Accessing the file storage system

The following code is extracted from the class "com.YEdit.Interface.FactoryCreate" to demonstrate how the correct storage system is loaded, and to show that it returns a class that conforms to the Java Interface "IAccess", which is the middle interface, this is shown in the next appendix.

```
public static final IAccess access(String type, USL fileObject)
    {
    String classNamePrefs = null;
    MasterPreference prefs;
    IAccess file = null;
    StringBuffer className = new StringBuffer("FactoryAccess.");
    className.append(type);
```

Sets up the variables and creates the name of the preference that holds the correct class to load (for example on YEdit.com "File.YEdit" is passed into this method).

```
    try {
        // Only load the default, system, and user system preferences
        prefs = MasterPreference.instance();

        classNamePrefs = prefs.getPreference(className.toString());
        // Check that it was found
        if( null == classNamePrefs )
            {
            Log.fatalError("Error: Could not find '" + className.toString() +
"' in the system preferences.");
            }
```

The code above retrieves the name of the class that is used for the type of file storage system passed. Once that is known, an object is created that contains that class, which is set up with the name of the file to use, and then the object is passed back to the calling code. The object that is returned uses the Java Interface "IAccess" to give access to the internal methods.

```
        file = (IAccess)Class.forName(classNamePrefs).newInstance();
        file.setUSL(fileObject);
        }
    catch(Exception e)
        {
        Log.error(e, "ERROR: Instantiating " + classNamePrefs + " from
factory.");
        file = null;
        }
    return( file );
    }
```

7.3 Appendix C: The Middle Interface

The middle interface has two parts. The first part is the Java Interface that is called "com.YEdit.Interface.IAccess". This Java Interface defines the methods that are implemented in order to allow access to the server interface for support of the requests that may be made. The second part is the super class called "com.YEdit.Interface._Access". This is an optional super class for any of the access factory classes. This super class assigns default behaviour to many of the methods and it allows one place for code that may be repeated in each sub-class.

com.YEdit.Interface

```
public abstract interface IAccess
```

An interface that defines the methods that are implemented in order to assure and enable consistent access to them. These classes **must** be able to stand-alone, that is they only rely on standard classes that are guaranteed to be present.

Copyright: Copyright (c) Daniel Hean 1998-2000, all rights reserved.
Date: 7 March 2000
Version: alpha 0.2000.03.7
Author: Daniel Hean

Method Detail

```
public java.lang.String[] list()
```

Returns a list of the files in the directory specified by this object.

```
public java.lang.String[] list(java.io.FilenameFilter filter)
```

Returns a list of the files in the directory specified by this object that satisfy the specified filter.

```
public boolean mkdir()
```

Creates a directory whose path name is specified by this object.

```
public boolean mkdirs()
```

Creates a directory whose path name is specified by this object, including any necessary parent directories.

```
public void setUSL(com.Inishgold.USL.USL valueUSL)
```

Set the internal value that stores the information source that we are currently looking at.

```
public com.Inishgold.USL.USL getUSL()
```

Return the USL of the current file.

```
public long getExpiration()
```

Returns the value of the expires header field.

```
public long lastModified()
```

Returns the time that the object was last modified.

```
public java.lang.String lastModifiedBy()
```

Returns who last modified the object.

```
public long length()
```

Returns the length of the file represented by this object.

```
public java.lang.String getReadWriteAccess()
```

Returns the read, write and execute permissions for this file.

```
public boolean exists()
```

Tests if this File exists.

```
public boolean canRead()
```

Tests if the application can read from the specified file.

```
public boolean canWrite()
```

Tests if the application can write to this file.

```
public com.Inishgold.USL.IVersion highestRevision()
```

Return the latest revision number for this file (Do **not** cache, unless all possible changes are caught).

```
public boolean lock(java.lang.String userName, java.lang.String location)
    throws java.io.IOException
```

Lock the object so that it is possible to read from and update the information pointed to by the object. This allows only one person to have an open output stream at any time.

```
public void unlock()
```

Release the write lock and close the stream, so that others can write to it and by closing the stream, it ensures that we can't mistakenly write to it again, even if we have stored the stream.

```
public boolean isLocked()
```

Check to see if the object is currently locked.

```
public java.lang.String getLockedWhenStr()
```

Retrieve the time that this was locked (in String format).

```
public java.io.InputStream getInputStream()
    throws java.io.IOException
```

Retrieve the stream to use for input from this resource.

```
public java.io.OutputStream getOutputStream()
    throws java.io.IOException
```

If the output stream is locked this will return the stream to use, if it is not currently locked, this **must** throw an IOException. Note: When the stream is unlocked, the stream that is given by this method **will** be closed, so as to avoid unlocked writes.

```
public java.io.OutputStream getErrorStream()
    throws java.io.IOException
```

Retrieve the stream for errors related to this object, this should always be available.

```
public java.io.PrintWriter getErrorWriter()
    throws java.io.IOException
```

Retrieve the PrintWriter for errors related to this object, this should always be available.

```
public void print(java.io.InputStream in)
    throws java.io.IOException
```

Write the contents of the supplied InputStream to the current object's OutputStream.

```
public void print(java.lang.String in)
    throws java.io.IOException
```

Write the contents of the supplied String to the current object's OutputStream.

```
public void println(java.lang.String in)
    throws java.io.IOException
```

Write the contents of the supplied String to the current object's OutputStream adding a newline to the end.

```
public boolean delete()
```

Deletes the file specified by this object.

```
public boolean renameTo(com.Inishgold.USL.USL name)
```

Renames the file specified by this object to have the path name given by the supplied argument.

com.YEdit.Interface

```
public abstract class _Access
     extends java.lang.Object
     implements IAccess
```

The _Access class is the optional super class of any access factory classes. It is optional to use this class as a super class for the access factory classes because all the required methods are defined in the interface (IAccess). The reason to use this as a super class is to organise in one place the methods that are repeatedly written in each access factory class.

Copyright: Copyright (c) Daniel Hean 1998-2000, all rights reserved.
Date: 17 May 2000
Version: alpha 0.2000.05.17
Author: Daniel Hean

Constructor Detail

```
protected _Access()
     throws java.io.IOException
```

Method Detail

```
public boolean isLocked()
```

Check to see if the object is currently locked.
Specified by: isLocked in interface IAccess

```
protected abstract boolean setLockedWhoLoc(java.lang.String who,
     java.lang.String location)
```

Set the name of the person who has this resource locked for writing. If the underlying resource is locked, then return false to signify that the lock failed. This must **never** be called directly, not even by this class.

```
public abstract java.lang.String getLockedWho()
```

Retrieve the name of the person who has this resource locked for writing.

```
public abstract long getLockedWhen()
```

Retrieve the time that this was locked (in Unix format).

```
public abstract java.lang.String getLockedWhenStr()
```

Retrieve the time that this was locked (in String format).
Specified by: getLockedWhenStr in interface IAccess

```
public boolean lock(java.lang.String who, java.lang.String location)
     throws java.io.IOException
```

Lock the object so that it is possible to read from and update the information pointed to by the object. This allows only one person to have an open output stream at any time.
Specified by: lock in interface IAccess

```
public void unlock()
```

Release the write lock and close the stream, so the others can write to it and by closing the stream, it ensures that we can't mistakenly write to it again, even if we have stored the stream.
Specified by: unlock in interface IAccess

```
public com.Inishgold.USL.USL getUSL()
```

Return the USL of the file.
Specified by: getUSL in interface IAccess

```
public void setUSL(com.Inishgold.USL.USL valueUSL)
```

Set the internal value that stores the information source that we are currently looking at.
Specified by: setUSL in interface IAccess

```
public void print(java.io.InputStream in)
    throws java.io.IOException
```

Write the contents of the supplied InputStream to the current object's OutputStream.
Specified by: print in interface IAccess

```
public void print(java.lang.String in)
    throws java.io.IOException
```

Write the contents of the supplied String to the current object's OutputStream.
Specified by: print in interface IAccess

```
public void println(java.lang.String in)
    throws java.io.IOException
```

Write the contents of the supplied String to the current object's OutputStream adding a newline to the end.
Specified by: println in interface IAccess

```
protected abstract java.io.InputStream
    setInputStream(com.Inishgold.USL.USL file)
    throws java.io.IOException
```

This is to be overridden in the subclass to do the actual creation of the input stream. This should return this class's inputStream.

```
protected abstract java.io.OutputStream
    setOutputStream(com.Inishgold.USL.USL file)
    throws java.io.IOException
```

This is to be overridden in the subclass to do the actual creation of the output stream. This should return this class's output stream.

```
protected abstract java.io.OutputStream setErrorStream()
    throws java.io.IOException
```

This is to be overridden in the subclass to do the actual creation of the error stream. This should return this class's error stream, and it **must** be available.

```
public java.io.InputStream getInputStream()
    throws java.io.IOException
```

Retrieve the stream to use for input from this resource.
Specified by: getInputStream in interface IAccess

```
public java.io.OutputStream getOutputStream()
    throws java.io.IOException
```

If the output stream is locked this will return the stream to use, if it is not currently locked, this **must** throw an IOException. Note: When the stream is unlocked, the stream that is given by this method **will** be closed, so as to avoid unlocked writes.
Specified by: getOutputStream in interface IAccess

```
public java.io.OutputStream getErrorStream()
    throws java.io.IOException
```

Retrieve the stream for errors related to this object, this should always be available.
Specified by: getErrorStream in interface IAccess

```
public java.io.PrintWriter getErrorWriter()
    throws java.io.IOException
```

Retrieve the PrintWriter for errors related to this object, this should always be available.
Specified by: getErrorWriter in interface IAccess

```
public boolean exists()
```

Tests if this file exists.
Specified by: exists in interface IAccess

```
public boolean canRead()
```

Tests if the application can read from the specified file.
Specified by: canRead in interface IAccess

```
public boolean canWrite()
```

Tests if the application can write to this file.
Specified by: canWrite in interface IAccess

```
public long lastModified()
```

Returns the time that the object was last modified.
Specified by: lastModified in interface IAccess

```
public java.lang.String lastModifiedBy()
```

Returns who last modified the object.
Specified by: lastModifiedBy in interface IAccess

```
public long getExpiration()
```

Returns the value of the expires header field.
Specified by: getExpiration in interface IAccess

```
public long length()
```

Returns the length of the file represented by this object.
Specified by: length in interface IAccess

```
public java.lang.String getReadWriteAccess()
```

Returns the read, write and execute permissions for this file.
Specified by: getReadWriteAccess in interface IAccess

```
public com.Inishgold.USL.IVersion highestRevision()
```

Return the latest revision number for this file (Do **not** cache, unless all possible changes are caught).
Specified by: highestRevision in interface IAccess

```
public boolean delete()
```

Deletes the file specified by this object.
Specified by: delete in interface IAccess

```
public boolean renameTo(com.Inishgold.USL.USL name)
```

Renames the file specified by this object to have the path name given by the supplied argument.
Specified by: renameTo in interface IAccess

```
public java.lang.String[] list()
```

Returns a list of the files in the directory specified by this object.
Specified by: list in interface IAccess

```
public java.lang.String[] list(java.io.FilenameFilter filter)
```

Returns a list of the files in the directory specified by this object that satisfy the specified filter.
Specified by: list in interface IAccess

```
public boolean mkdir()
```

Creates a directory whose path name is specified by this object.
Specified by: mkdir in interface IAccess

```
public boolean mkdirs()
```

Creates a directory whose path name is specified by this object, including any necessary parent directories.
Specified by: mkdirs in interface IAccess

7.4 Appendix D: The Server Interface (File System example)

com.YEdit.ServerInterface.FileSystem

```
public class FileSystem
    extends _Access
    implements IAccess
```

The FileSystem class is the factory access class that controls access to the systems file system.

Note: The reason that this does **not** subclass the applicable File/JDBC classes is because that would expose methods and variables that are not consistent across those different classes. That would mean that these classes would no longer be modular as they have been designed, and you would no longer be able to just plug in the one you want and use it.

Copyright: Copyright (c) Daniel Hean 1998-2000, all rights reserved.
Date: 17 Sept 2000
Version: alpha 0.2000.09.17
Author: Daniel Hean

Constructor Detail

```
public FileSystem()
    throws java.io.IOException
```

Method Detail

```
protected boolean setLockedWhoLoc(java.lang.String who,
    java.lang.String location)
```

Set the name of the person who has this resource locked for writing. If the underlying resource is locked, then return false to signify that the lock failed. This must **never** be called directly, not even by this class.
Overrides: setLockedWhoLoc in class _Access

```
public java.lang.String getLockedWho()
```

Retrieve the name of the person who has this resource locked for writing.
Overrides: getLockedWho in class _Access

```
public java.lang.String getLockedLocation()
```

Retrieve the location that this was locked from.

```
public long getLockedWhen()
```

Retrieve the time that this was locked (in Unix format).
Overrides: getLockedWhen in class _Access

```
public java.lang.String getLockedWhenStr()
```

Retrieve the time that this was locked (in String format).
Specified by: getLockedWhenStr in interface IAccess
Overrides: getLockedWhenStr in class _Access

```
protected java.io.InputStream setInputStream(com.Inishgold.USL.USL file)
    throws java.io.IOException
```

Return the file stream to use for input from this file.
Overrides: setInputStream in class _Access

```
protected java.io.OutputStream setOutputStream(com.Inishgold.USL.USL file)
    throws java.io.IOException
```

Return the file stream to use for output to this file.
Overrides: setOutputStream in class _Access

```
protected java.io.OutputStream setErrorStream()
    throws java.io.IOException
```

Return the file stream to use for output to this file's error channel.
Overrides: setErrorStream in class _Access

```
public boolean exists()
```

Tests if this file exists.
Specified by: exists in interface IAccess
Overrides: exists in class _Access

```
public boolean canRead()
```

Tests if the application can read from the specified file.
Specified by: canRead in interface IAccess
Overrides: canRead in class _Access

```
public boolean canWrite()
```

Tests if the application can write to this file.
Specified by: canWrite in interface IAccess
Overrides: canWrite in class _Access

```
public com.Inishgold.USL.IVersion highestRevision()
```

Return the latest revision number for this file (Do **not** cache, unless all possible changes are caught).
Specified by: highestRevision in interface IAccess
Overrides: highestRevision in class _Access

8 References

[Alexander 1977] Alexander C. (1977). *"A Pattern Language: Towns, Buildings, Construction"*. Oxford University Press

[Alexander 1979] Alexander C. (1979), *"The Timeless Way of Building"*. Oxford University Press

[Appelt 1999] Appelt, W. (1999). "WWW Based Collaboration with the BSCW System", in *Proceedings of SOFSEM'99*, Springer Lecture Notes in Computer Science 1725, p.66-78; November 26 - December 4, Milovy (Czech Republic)

[BSCW] Basic Support for Co-operative Work (BSCW) enables collaboration over the Web. BSCW is a 'shared workspace' system which supports document uploads, event notification, group management and much more. To access a workspace you only need a standard Web browser. http://bscw.gmd.de/ (current 30th November 2000)

[Bentley Horstmann Trevor 1997] Bentley, R., Horstmann, T., Trevor, J. (1997). "The World Wide Web as enabling technology for CSCW: The case of BSCW", in *Computer-Supported Cooperative Work: Special issue on CSCW and the Web*, Vol. 6, Kluwer Academic Press.

[Berners-Lee 1989] Berners-Lee, T. (March 1989), "Information Management: A Proposal"
See [Berners-Lee WtW] and http://www.w3.org/History/1989/proposal.html (current 30th November 2000) W3 Status: "Provided for historical interest only"

[Berners-Lee 1990] Berners-Lee, T. (1990). "Intended uses for hypertext at CERN" http://www.w3.org/DesignIssues/Uses.html (current 30th November 2000) W3 Status: W3 Archive

Examples of the original intended uses for the Web:

General Reference/Encyclopaedia

Encyclopaedia Britannica http://www.britannica.com/ (current 30th November 2000)

Merriam-Webster's Collegiate Dictionary http://www.m-w.com/ (current 30th November 2000)

Centralised publishing (online help/documentation/tutorials/etc)

FAQs http://www.faqs.org/ (current 30th November 2000)

News sites

TVNZ http://onenews.nzoom.com/ (current 30th November 2000)

CNN http://www.cnn.com/ (current 30th November 2000)

BBC http://www.bbc.co.uk/ (current 30th November 2000)

Personal notebooks/homepages

Geocities http://geocities.yahoo.com/ (current 30th November 2000)

Tripod http://www.tripod.lycos.com/ (current 30th November 2000)

Angelfire http://angelfire.lycos.com/ (current 30th November 2000)

Collaborative authoring

BSCW http://bscw.gmd.de/ (current 30th November 2000)

[Berners-Lee 1991] Berners-Lee, T. (1991-1992). *"W3 Concepts"* See http://www.w3.org/Talks/General/Concepts.html (current 30th November 2000) W3 Status: "This talk is an old one from 1991 and 1992, out of date, though interesting for historical reasons"

[Berners-Lee WtW] Berners-Lee, T. (1999). *"Weaving the Web"*, HarperSanFrancisco. ISBN: 0-06-251586-1

[Booch 1986] Booch, G. (Feburary 1986). "Object-Oriented Development", *IEEE Trans. Software Engineering*, pages 211-221

[Bush 1945] Bush, V; science adviser to President Roosevelt. (July 1945). "As we may think", (*The Atlantic Monthly*; July 1945; Volume 176, No 1; pages 101-108) http://www.w3.org/History/1945/vbush/ (current 30th November 2000)

[Cailliau 1995] Cailliau, R. (November 1995). "A Short History of the Web", Text of a speech delivered at the launching of the European branch of the W3 Consortium in Paris. http://www.inria.fr/Actualites/Cailliau-fra.html (current 30th November 2000).

[Comerford 1999] Comerford, R. (May 1999), "The path to Open-Source systems", *IEEE Spectrum*, pages 25-31

[CVS] *Concurrent Versions System* http://www.cvshome.org/ (current 30th November 2000)

[Cunningham] *WikiWiki*. (Cunningham creator/maintainer), http://c2.com/cgi-bin/wiki (current 30th November 2000).
Quote from their front page: "(It is) ... a fun way of communicating asynchronously across the network", or as a set of web pages that are open and free for anyone to edit as they wish. Wiki is not real-time; therefore people have time to think before they follow up a web page, often days or weeks, so they have time to consider what they write.

[Dominus 1998] Dominus, M. (January 1998). "Perl: Not just for Web programming", *IEEE Software*, pages 69-74

[Fogel 2000] Fogel K. (2000). *Open Source Development with CVS*. Coriolis Inc. ISBN: 1576104907 http://cvsbook.red-bean.com/ (current 30th November 2000)

[Fowler 1999] Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Pub Co. ISBN: 0201485672

[Hecker 1999] Hecker, F. (January 1999). "Setting up shop: The business of Open-Source software", *IEEE Software*, pages 45-51

[Java] Gosling, J & McGilton, H (May 1996). *The Java Language Environment: A White Paper*, http://java.sun.com/docs/white/langenv/ (current 30th November 2000).

[Merriam Webster] (1999). *Merriam-Webster's Collegiate Dictionary.* Tenth Edition. Merriam-Webster, Incorporated

[Nelson 1981] Nelson, T. (1981). *Literary Machines.* ISBN: 089347052X

[Netcraft 2000] Netcraft web server survey in September 2000 http://www.netcraft.com/survey/ (current 30th November 2000). There were 21,166,912 web hosts when Netcraft conducted this survey in September 2000.

[NetValue 2000] NetValue http://www.netvalue.com/ (current 30th November 2000) has set up a market research system in Europe and in the United-States.

[Pressman 1992] Pressman, R (1992). *Software Engineering, a practitioner's approach.* (Third edition). McGraw-Hill. ISBN: 0-07-112779-8

[PRCS] *Project Revision Control System* http://www.xcf.berkeley.edu/~jmacd/prcs.html (current 30th November 2000)

[RFC.... / STD ....] Request for comment and Internet Standards RFCs and STDs are officially available from either the IETF or http://www.rfc-editor.org/ (current 30th November 2000), which also lists other repositories.

HTTP [RFC 2616]

WebDAV [RFC2518]

HTML [RFC2854]

Telnet [RFC854-855 / STD8]

FTP [RFC959 / STD9]

IRC [RFC2810-3]

Email SMTP [RFC821 / STD10]

POP3 [RFC1939 / STD53]

Usenet (NNTP) [RFC977/RFC1036]

Gopher [RFC1436]

WAIS [RFC1625]

[RCS] *Revision Control System* http://www.linuxdoc.org/HOWTO/mini/RCS-1.html (current 30th November 2000)

[Risks] *Comp Risks.* (Neumann, P moderator). http://catless.ncl.ac.uk/Risks/, news:comp.risks (current 30th November 2000).

[Santayana 1905] Santayana, G (1863-1952). US philosopher, author, educator, poet. "Those who cannot remember the past, are condemned to repeat it." *Life of Reason*, "Reason in Common Sense" ch. 12 (1905-6)

[Servlets] "Servlets are server side Java programs that extend a web server in a similar fashion to CGI scripts", http://java.sun.com/products/servlet/whitepaper.html (current 30th November 2000)

[SourceSafe] *SourceSafe* http://www.microsoft.com/ (current 30th November 2000)

[Whittaker 2000] Whittaker, J. (January 2000). "What is software testing? And why is it so hard?", *IEEE Software*, pages 70-79

[W3 1995] World Wide Web Consortium. *"A Little History of the World Wide Web"*. http://www.w3.org/History.html (current 30th November 2000) W3 Status: For Archival/Historical Interest