

Examensarbete


**Collaborative web content management
- Wiki and beyond**

av

Mattias Beermann

LITH-IDA-EX--05/087--SE

2005-12-08

| | | |
|---|--|--|
| Avdelning, institution Division, department Institutionen för datavetenskap Department of Computer and Information Science | Datum Date 2005-12-08 |  Linköpings universitet |
|---|--|--|

| | | |
|--|--|---|
| Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____ | Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport _____ | ISBN _____ ISRN LITH-IDA-EX--05/087--SE Serietitel och serienummer ISSN _____ Title of series, numbering |
|--|--|---|

URL för elektronisk version

Titel
 Title
 Collaborative web content management - Wiki and beyond

Författare
 Author
 Mattias Beermann

Sammanfattning
 Abstract

Wiki web sites are collaborative content management systems where everything can be edited by anyone, a concept that at first glance seems to be impossible due to vandalism and spam. Wikipedia.org contains more than one million articles, all of them are editable by anyone. Thanks to peer reviewing and tools that enable an article to be reverted to a previous version, vandalism and spam can be controlled efficiently. The wiki concept has some challenges ahead, to be able to handle the rapid growth, to standardize the markup language used to write articles and to create better editors that can be used by anyone without any special markup language knowledge.

This thesis provides an extensive background to the wiki concept and possible solutions to the above problems. A wiki XML language is designed, that is simple, extensible and uses some of the solutions proposed in the XHTML 2.0 draft recommendation. Different solutions are proposed for a browser based WYSIWYG XML editor together with experiences from an experimental implementation. Architecture and design considerations for a scalable and high performance wiki engine are described and experiences from a C# 2.0 wiki engine implementation, code named KuaiWiki, are presented.

Nyckelord
 Keywords
 Wiki, Web Content Management, C#, XML, KuaiWiki

Examensarbete

**Collaborative web content management
- Wiki and beyond**

av

Mattias Beermann

LITH-IDA-EX--05/087--SE

2005-12-08

Handledare: Adrian Pop

Examinator: Peter Fritzson

Abstract

Wiki web sites are collaborative content management systems where everything can be edited by anyone, a concept that at first glance seems to be impossible due to vandalism and spam. Wikipedia.org contains more than one million articles, all of them are editable by anyone. Thanks to peer reviewing and tools that enable an article to be reverted to a previous version, vandalism and spam can be controlled efficiently. The wiki concept has some challenges ahead, to be able to handle the rapid growth, to standardize the markup language used to write articles and to create better editors that can be used by anyone without any special markup language knowledge.

This thesis provides an extensive background to the wiki concept and possible solutions to the above problems. A wiki XML language is designed, that is simple, extensible and uses some of the solutions proposed in the XHTML 2.0 draft recommendation. Different solutions are proposed for a browser based WYSIWYG XML editor together with experiences from an experimental implementation. Architecture and design considerations for a scalable and high performance wiki engine are described and experiences from a C# 2.0 wiki engine implementation, code named KuaiWiki, are presented.

The conclusions are:

- The wiki concept will continue to grow in popularity.
- XML is suitable as a markup language used to specify wiki articles together with XSLT for transformations.
- Browser based WYSIWYG XML editors are possible, but hard to implement due to limitations in the browsers.
- A wiki engine can use XML and XSLT to present articles. By using the new functions in Microsoft SQL Server 2005 some of the processing can take place inside the database.
- An implementation that is from the ground up designed for performance and scalability should be able to handle large wiki web sites and wiki hosting scenarios. The use of caching at several levels in the application can greatly enhance the performance.

Contents

| | | |
|----------|--|----------|
| 1 | Introduction | 1 |
| 1.1 | That cannot work! | 2 |
| 1.2 | Reasons to consider wiki | 2 |
| 1.3 | Different sites, different needs | 3 |
| 1.4 | Wiki editing | 3 |
| 1.5 | Purpose of the thesis | 4 |
| 2 | Wiki evolution | 6 |
| 2.1 | History | 6 |
| 2.2 | Susning.nu | 6 |
| 2.3 | Wikimedia | 7 |
| 3 | Review of wiki features | 9 |
| 3.1 | Large variety | 9 |
| 3.2 | FlexWiki | 10 |
| 3.2.1 | Features | 10 |
| 3.3 | MediaWiki | 12 |
| 3.4 | Look and feel | 12 |
| 3.4.1 | Skins | 12 |
| 3.4.2 | Readability | 13 |
| 3.4.3 | Tool tips | 13 |
| 3.4.4 | Side bar | 13 |
| 3.4.5 | Stub threshold | 13 |
| 3.4.6 | Printable version | 14 |
| 3.4.7 | Report style | 14 |
| 3.5 | Multimedia | 15 |
| 3.6 | Version tracking | 15 |
| 3.6.1 | Diff | 15 |
| 3.6.2 | Edit notification | 15 |
| 3.6.3 | Backup | 16 |
| 3.7 | Page types | 16 |
| 3.8 | Editing | 16 |
| 3.8.1 | Smart tags | 17 |
| 3.8.2 | Multilanguage support | 17 |
| 3.9 | Search and queries | 17 |
| 3.10 | Spam and vandalism protection | 17 |
| 3.11 | Extensibility | 17 |

| | | |
|----------|--|-----------|
| 4 | Document formats | 18 |
| 4.1 | Current situation | 18 |
| 4.1.1 | Different dialects of wiki text | 18 |
| 4.1.2 | HTML and XHTML | 19 |
| 4.2 | Markup Requirements | 20 |
| 4.3 | Existing popular markup languages | 21 |
| 4.3.1 | LaTeX | 21 |
| 4.3.2 | DocBook | 22 |
| 4.3.3 | (X)HTML | 22 |
| 4.3.4 | XML | 24 |
| 4.4 | WXML - a wiki XML markup language | 25 |
| 4.4.1 | Style | 25 |
| 4.4.2 | Modularization of XHTML | 26 |
| 4.4.3 | Metadata and article markup | 27 |
| 4.4.4 | Core modules | 27 |
| 4.4.5 | Example of a WXML document | 28 |
| 4.5 | Transformation of XML documents | 29 |
| 4.5.1 | Overview of XSLT | 29 |
| 5 | Wiki article editing | 31 |
| 5.1 | Textarea XML Editing | 31 |
| 5.2 | Wiki text editing | 32 |
| 5.3 | WYSIWYG XML editing | 32 |
| 5.4 | Experimental implementation | 32 |
| 5.4.1 | HTML WYSIWYG editors | 32 |
| 5.4.2 | Different approaches | 33 |
| 5.4.3 | Lack of XML support | 33 |
| 5.4.4 | Cut and paste | 33 |
| 5.4.5 | DTD/Schema awareness | 34 |
| 5.4.6 | Browser issues | 34 |
| 5.4.7 | Future work | 35 |
| 5.5 | Multiple required editors | 35 |
| 6 | Architecture and design considerations | 37 |
| 6.1 | Performance objectives | 38 |
| 6.1.1 | Wikipedia.org as a performance example | 38 |
| 6.2 | Deployment scenarios | 39 |
| 6.2.1 | Wiki on a stick | 39 |
| 6.2.2 | Personal wiki server | 41 |
| 6.2.3 | Wiki hosting | 41 |
| 6.2.4 | Summary | 41 |
| 6.3 | Client scenarios | 42 |
| 6.3.1 | Fat client | 42 |
| 6.3.2 | Slim clients | 44 |
| 6.3.3 | Wiki web service | 44 |
| 6.3.4 | Search robots | 45 |
| 6.4 | Programming language | 45 |
| 6.4.1 | Java | 45 |
| 6.4.2 | C# | 46 |
| 6.4.3 | Performance | 47 |

| | | |
|----------|--|-----------|
| 6.4.4 | Choice of programming language | 47 |
| 6.5 | Storage system | 48 |
| 6.5.1 | Performance of database servers | 48 |
| 6.5.2 | MS SQL Server 2005 | 49 |
| 6.6 | Performance versus simplicity | 50 |
| 6.6.1 | Reflector for .NET | 51 |
| 6.7 | Scale out versus Scale up | 51 |
| 6.7.1 | Scale up | 51 |
| 6.7.2 | Scale out | 52 |
| 6.7.3 | Loosely coupled and layered design | 52 |
| 6.8 | Design patterns | 52 |
| 6.8.1 | Adapter pattern | 52 |
| 6.8.2 | Provider pattern | 53 |
| 6.9 | Caching | 54 |
| 6.9.1 | Database cache dependency | 54 |
| 6.9.2 | IIS and ASP.NET | 54 |
| 6.10 | CSS stylesheets | 55 |
| 6.11 | XSL stylesheets | 55 |
| 6.12 | Preprocess | 55 |
| 6.12.1 | Web server level caching | 56 |
| 6.12.2 | Database level caching | 56 |
| 6.13 | Database optimization | 56 |
| 6.13.1 | Table layout | 56 |
| 6.13.2 | Indexes | 57 |
| 6.13.3 | Stored procedures | 57 |
| 6.13.4 | XML Storage | 57 |
| 6.13.5 | Replication | 58 |
| 7 | Implementation | 59 |
| 7.1 | Prototype in .NET Framework 1.1 | 59 |
| 7.2 | .NET Framework 2.0 | 60 |
| 7.2.1 | SQL Server 2005 | 60 |
| 7.3 | Modified provider pattern | 60 |
| 7.4 | Processing steps | 62 |
| 7.4.1 | Request class | 63 |
| 7.4.2 | WebConfig | 63 |
| 7.4.3 | Pipeline | 63 |
| 7.4.4 | KuaiWikiMainPageSource | 66 |
| 7.4.5 | InterceptHandler | 67 |
| 7.4.6 | InterceptingXmlReader | 67 |
| 7.4.7 | xPathNavigableConverter | 68 |
| 7.5 | Database | 69 |
| 7.5.1 | Table layout | 69 |
| 7.5.2 | Stored Procedures | 70 |
| 7.5.3 | Triggers | 73 |
| 7.5.4 | Functions | 75 |
| 7.6 | Future work | 77 |

| | | |
|----------|--------------------------------------|-----------|
| 8 | Conclusion | 78 |
| 8.1 | Document formats | 78 |
| 8.2 | WYSIWYG Editing | 78 |
| 8.3 | Wiki engine implementation | 79 |
| 8.4 | The wiki concept | 80 |

Chapter 1

Introduction

The basic idea behind the wiki concept is simple. The inventor is Ward Cunningham and he originally described it as:

The simplest online database that could possibly work. (Leuf and Cunningham, 2001)

The current largest wiki web is Wikipedia.org, they describe wiki as:

...a group of Web pages that allows users to add content, as on an Internet forum, but also allows others (often completely unrestricted) to edit the content. (Wikipedia: Wiki, 2005-11-08)

The core concept of wiki is just that, it allows users to add content, but it also allows anyone to edit that content. It is common that web sites have some sort of functionality for users to add comments to pages, but wiki also enables users to change the content, and not just the content the user himself has written, but all content.

The names originate from the Hawaiian term wiki that means quick or super-fast.

The essence of wiki is *less is more*:

- Simple navigation
- Edit, a click away
- Dirt simple markup
- Anyone can change anything
- Fast retrieval
- Built in search
- Quick cross linking
- Encourage linking
- No broken links
- Links are page titles

(Leuf and Cunningham, 2001)

1.1 That cannot work!

When people first hear about the wiki concept, the first response is usually *that cannot work!* Wikipedia, the largest online encyclopedia with more than 800 000 articles just in English, and even more articles in other languages, is a wiki site, editable by anyone, and it is becoming larger and larger, and more and more popular.

The objection most people have against the wiki concept is that it must be impossible to avoid vandalization, that people will erase content, or even worse, add content that is not true. All this happens to a wiki, and can sometimes be a large problem, but all articles are peer reviewed, often very quickly, and anyone that finds vandalism, or a removed article, or content that is not true, can easily revert to the last proper version of that article. IBM has studied vandalism on Wikipedia and most cases of vandalism were found to be reverted in five minutes or less (Wikipedia: Wiki, 2005-11-08). There are multiple ways to get informed when an article is changed: it can be placed on a special watch list, an e-mail with the changes can be sent or RSS-feeds and IRC-channels can be used, to just name a few.

There is nothing that stops a person to publish a web page on an ordinary web server, writing facts that are not true, and nothing stops him from specifying a well known person's name as the author. Are the facts on such a page truer, more believable than on a page that anyone can change and correct? Just because people can not vandalize a normal web page, there is nothing ensuring that the information on that page is correct, and there is usually no way for people to give feedback, at least not feedback that is available to the next visitor of that specific page.

On a wiki site, this is possible. If it is obvious that the content is not true, anyone can revert back to a previous version of the article, or if the visitor is unsure about something, he or she can add a note about this on the corresponding discussion page.

Most wikis have discussion pages, for each wiki article, there is a discussion page. This page is for discussion about the article, about doubts in the content, or just to ask others if it is a good idea to add some information about this or that. The discussion page itself is also a wiki page, so everyone can edit anything on it. Usually the discussion page is longer than the corresponding wiki article and gets more frequent updates.

1.2 Reasons to consider wiki

There are many different wiki engines and most of them can be used for free. The wiki concept can have many types of information and uses, some of them are:

- Free form notebook, log book, brainstorming
- Address book and resource finder, link library
- Informal register applications, videos, books, photo albums
- Document manager, linking to document on disk

- Resource collections
- Collaborative FAQs
- Project management
- Web site management
- Discussion and review
- Shared bulletin board posting
- Online guestbook
- Free-form database

(Leuf and Cunningham, 2001)

The wiki engines enable easy editing of articles without any required HTML knowledge. Anyone can edit the information, the quick editing and the discussion pages enable quick collaboration and updates. There are wiki engines for most platforms, and wiki engines adapted to most usage scenarios.

1.3 Different sites, different needs

A Wiki site is not a solution that fits all types of content. There is probably not a good idea for a company to have their company presentation and product pages as a public wiki site that anyone can edit.

Not everyone needs a wiki. Not everyone wants a wiki. Not every situation benefits from becoming an open discussion or collaboration forum. (Leuf and Cunningham, 2001)

The companies' intranet, restricted to people working at the company, is probably a better place for a wiki, having a collaborative discussion about products, customer complains etc.

Wiki is not a solution that fits all web site needs, but for some types of information it is a very powerful tool to get people to collaborate and create content that would be impossible for a single user, or a single company to write. Wikipedia is one proof of many that the wiki concept works, and that wiki can make people contribute to create large amount of valuable content.

1.4 Wiki editing

The editing of wiki articles is usually performed on a markup language called wiki text. Due to the evolution of many different wiki engines, and the rapid development, there are many different wiki text dialects that usually share some basic tags, but otherwise are incompatible. To continue the success of wikis, it is probably necessary to create a standardized wiki text language, or to switch to visual editing.

The wiki text language was designed to be very compact, easy to learn, and easy to write. Wiki text is mostly focused at marking up structure, and the editor has limited control of the layout.

You're browsing a database with a program called WikiWikiWeb. And the program has an attitude. The program wants everyone to be an author. So, the program slants in favor of authors at some inconvenience to readers. (Leuf and Cunningham, 2001)

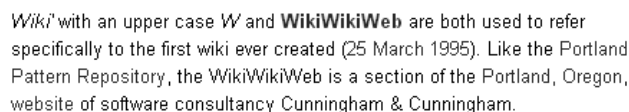
An example of wiki text markup in the MediaWiki dialect:

```
\texttt{'Wiki' with an upper case 'W' and ''[[WikiWikiWeb]]' are both used to refer specifically to the first wiki ever created ([[25 March]], [[1995]]). Like the [[Portland Pattern Repository]], the WikiWikiWeb is a section of the [[Portland, Oregon]], [[website]] of software consultancy Cunningham & Cunningham.}
```

This is transformed by the wiki engine into the following HTML code:

```
\texttt{<p><i>Wiki</i> with an upper case <i>W</i> and <b><a href="/wiki/WikiWikiWeb" title="WikiWikiWeb">WikiWikiWeb</a></b> are both used to refer specifically to the first wiki ever created (<a href="/wiki/March_25" title="March 25">25 March</a> <a href="/wiki/1995" title="1995">1995</a>). Like the <a href="/wiki/Portland_Pattern_Repository" title="Portland Pattern Repository">Portland Pattern Repository</a>, the WikiWikiWeb is a section of the <a href="/wiki/Portland_%2C_Oregon" title="Portland, Oregon">Portland, Oregon</a>, <a href="/wiki/Website" title="Website">website</a> of software consultancy Cunningham & Cunningham.</p>}
```

This is rendered by a web browser as:



Wiki with an upper case **W** and **WikiWikiWeb** are both used to refer specifically to the first wiki ever created (25 March 1995). Like the Portland Pattern Repository, the WikiWikiWeb is a section of the Portland, Oregon, website of software consultancy Cunningham & Cunningham.

Figure 1.1: Wiki text rendered in browser

The double apostrophes change the mode to italic, the triple apostrophes change the mode to bold and text inside double square brackets [[title of wiki article]] creates a link to that wiki article. There are wiki text tags to create lists of items, tables, timelines and many more structures describing structure or layout.

1.5 Purpose of the thesis

The wiki concept has over the last years grown in popularity, and there are now large amounts of high quality information available through wiki web sites. The growth and popularity have lead to problems. Wikis are now commonly the

target for vandalism and spam, both in the form of manual editing but also by special purpose bots.

The markup language most commonly used are based on different text tags, and most wikis employ manual editing of the documents without any WYSIWYG support. The editing creates a barrier for new, none technical users and the text markup is not standardized and is hard to validate and transform to other formats.

Wikipedia.org, the biggest wiki has large problems with performance. This might be because of the design and evolution of the MediaWiki engine that they use.

The purpose with this thesis is to investigate three main areas that are crucial for the ongoing success of wikis.

- Wiki markup as XML. How can the XML-format be used for wiki markup?
- Wiki document editing. How can WYSIWYG editing be used to improve the editing experience?
- Scalability and performance. How an implementation should be architected to have high scalability and performance?

For each area, a review of existing solutions will be done, highlighting the pros and cons. New solutions will be discussed and some of them will be implemented as a support for this thesis.

During the thesis project, the core of a modular wiki engine will be written and used to evaluate new solutions. The architecture and design of a modular scalable and high performing experimental wiki engine is the main focus of the thesis.

Chapter 2

Wiki evolution

The first wiki engine had only a few simple features, but that was ten years ago and today's top of the line wiki engines are advanced server applications including many advanced features.

This chapter describes the evolution of the wiki concept, from the beginning to present day, and also includes a discussion about future directions.

2.1 History

Ward Cunningham invented the wiki concept, implemented the first wiki engine, and started the first wiki web called WikiWikiWeb on March 25, 1995. Wiki is a Hawaiian term for quick, or fast, or to hasten. He learned the term from the wiki wiki shuttle buses at Honolulu Airport, instead of naming the web quick-web he chose the name wiki web.

The wiki concept became increasingly recognized in the late 1990s as a good way to develop knowledge based web sites. In the early 2000s enterprises started to use wiki webs for collaborative uses. Wikipedia was launched in January 2001. In 2002 Socialtext launched the first commercial open source wiki engine. By 2004, open source wikis such as MediaWiki, Kwiki and TWiki had over one million downloads reported on Sourceforge. Wikis are commonly used on corporate intranets, where cases of spam and vandalism become less of a problem. (Wikipedia: Wiki, 2005-11-08)

2.2 Susning.nu

In October 2001 susning.nu was opened, a Swedish wiki web started by Lars Aronsson. In less than a month the site had grown to include 2000 articles. At the end of 2001 the site contained 4000 pages (Aronsson, 2002-11-07), this has grown to present day when the site includes 58 000 articles (Susning.nu: Homepage, 2005-11-08).

Susning.nu has no license agreement which has the consequence that the contributions are copyrighted by their submitters. This makes all susning.nu content unavailable to use at other wiki sites without permission from the submitters. If Lars Aronsson decides to close down the site, the content can not be used somewhere else.

Susning.nu is using advertisements on their pages, something not found in wikimedia projects.

On the 15th of April 2004 the site was closed for editing due to large problems with vandalism, and since then it is only possible for a small group of editors to change the content. (Wikipedia: Susning.nu, 2005-11-08)

2.3 Wikimedia

The Wikimedia Foundation was started on June 20, 2003 by Wikipedia founder Jimmy Wales. It is the parent organization of the various Wikimedia projects. The foundation will maintain and develop free content in wiki based projects. The foundation owns the computer hardware, domain names and trademarks of all Wikimedia projects and the MediaWiki wiki engine. The content is licensed under the GNU Free Documentation License. (Wikimedia: About the Wikimedia Foundation, 2005-11-08)

Imagine a world in which every single person is given free access to the sum of all human knowledge. That's what we're doing. And we need your help. (Wikimedia: Home, 2005-11-08)

The different projects in the Wikimedia family are detailed below.

Wikipedia Free encyclopedias in all languages of the world. The wikipedia project is the largest in the wikimedia family of projects. Wikipedia now contains more than one million articles in well over 100 languages.

Wiktionary Free dictionaries and thesaurus in every language. Today the largest edition is the English, followed by Polish, Bulgarian and Dutch. The total number of entries is almost 200 000.

Wikiquote A repository of quotations, proverbs, mnemonics and slogans. In July 2005 it included nearly 18 000 pages in over 30 languages.

Wikibooks A collection of free e-books aimed at students and teachers at high-school and university level but there is also a Wikijunior section. The content include textbooks, language courses, manuals and annotated public domain books.

One example is a Chinese Mandarin course for English speakers. The advantage of books published as wikis are that the readers can give instant feedback to the authors and the content evolves over time.

Wikisource A collection of primary source texts, distributed as free and open content. It is a useful archive of classics, laws and other free texts.

Wikispecies A central extensive database for taxonomy aimed at scientific users. It will be a free directory of species, covering animalia, plantae, fungi, bacteria, archaea, protista and other forms of life.

Wikimedia Commons Provides a central repository of video, images, music and spoken texts to be used by other Wikimedia projects. In June 2005 it had over 150 000 multimedia files.

Wikinews One of the youngest projects, started in December 2004. This project will provide news article from a neutral point of view, ranging from original reports to summaries of news from external resources.

(Wikimedia: Our projects, 2005-11-08)

Chapter 3

Review of wiki features

The wiki concept has by now existed for ten years and over this period of time there have been many implementations of wiki engines. The simplicity of the concept makes a bare bone wiki engine implementable in a short timeframe. The advanced wiki engines used for larger sites contain a large number of functions and have become very advanced implementations.

This chapter begins with a quick overview of different wiki engines, and then the FlexWiki engine and the MediaWiki engine are discussed in some detail. The major part of this chapter is a feature list, describing the features that should exist in a modern general purpose wiki engine. The list is based on the feature list found in the MediaWiki manual but expanded with a lot more content and ideas (MediaWiki - Documentation: Introduction, 2005-11-11).

3.1 Large variety

The original and first Wiki, WikiWikiWeb, has a long list of different wiki implementations. The variety of implementation language is large, more than thirty programming languages are listed, everything from scripting languages, such as Classic ASP and PHP, to Ada, C++ and Java.

The languages with most implementations are PHP, Perl, Java and Python. PHP is the most popular of those, with around 70 different wiki engines. A lot of the engines are forks of other implementations. (WikiWikiWeb: Wiki Engines, 2005-11-11)

WikiWikiWeb also contains a top ten wiki list, the criteria used are best of class for a particular purpose, outstanding features and general popularity. The list is in no particular order. Remember that the list is a wiki page, and not the result of any formal voting, and the top ten list, contains only 9 items. (WikiWikiWeb: Top Ten Wiki Engines, 2005-11-11)

UseModWiki - implemented in Perl, and is based on Ward Cunningham's original WikiWiki implementation. Wikipedia used this engine before switching to their custom build MediaWiki engine.

PhpWiki - this engine is based on UseModWiki, but implemented in PHP and with many added features.

OddMuseWiki - another popular engine based on UseModWiki.

TwikiClone - a mature, stable and full featured Perl wiki engine developed for large corporate intranets. (WikiWikiWeb: Twiki Clone, 2005-11-11)

TikiWiki - a wiki trying to compete with content management systems. The implementation gets mixed reviews by users. (WikiWikiWeb: TikiWiki, 2005-11-11)

MediaWiki - this is the wiki engine used by Wikipedia and is one of the most popular wiki engines. It can handle large wiki sites and contains a rich set of features. The implementation is actively developed.

PmWiki - another implementation written in PHP, claims to be easy to install and have a simple design combined with a nice feature list.

WakkaWiki - featured on the Wiki Engine Hall of Fame, it has been split in at least seven different forks. The implementation is designed for speed and easy extensibility. (WikiWikiWeb: WakkaWiki, 2005-11-11)

MoinMoin - developed as a SourceForge project, implemented in Python and used by the Python language community at <http://www.python.org/moin> and many others. The implementation has a flexible and modular design, with separate parsing modules for different wiki text dialects. (WikiWikiWeb: Moin Moin, 2005-11-11)

3.2 FlexWiki

This is a wiki implementation written in the C# language. It was originally developed by David Ornstein working as a lead program manager in the digital documents group at Microsoft. He is still active in the project, but on the 27 September 2004 FlexWiki was released as an open source project at SourceForge.

He started FlexWiki as an experiment, he had been part of many projects where lack of communication and a shared vocabulary created problems. He thought that the wiki concept had the right features and was not too heavy weight as some other tools he had used. The goal of the FlexWiki implementation was to answer the question: could a good enough wiki implementation really improve the software development process at Microsoft?

In his blog post announcing the open source release, he is not still sure about the answer to this question, but several groups at Microsoft used FlexWiki at that time and he thought the project was ready to become an open source project.

FlexWiki is the third software released by Microsoft under their shared source initiative at SourceForge, the other two being WiX and WTL. These projects have become very successful, being in the top 5% of the most active projects at SourceForge.

FlexWiki was designed to allow multiple teams to work on related projects in the same wiki, with the help of separate namespaces. (Ornstein, 2005-11-11)

3.2.1 Features

The FlexWiki documentation has no up to date list of all the current features, but some of the features implemented are:

Easy editing - with the FormattingRules dialect of wiki text which is very similar to most wiki text implementations.

Email notification - based on subscription to WikiNewsletters. A WikiNewsletter is a collection of wiki articles. When any article contained in the WikiNewsletter is changed, every user subscribed to the newsletter will get an email with the changes.

RSS feeds - a feature that is very similar to the email notification, but the changes are published in an RSS feed.

Full article history - all versions of an article are saved. Functionality to see the changes between two versions is available. When referring to a wiki article, it is possible to refer to a specific version with an URL, creating a stable reference that will never change.

Article search - the possibility to search for an article based on its name and content. Regular expressions can also be used for searching.

Back links - contains a list of all the articles linking to the current page.

WikiTalk - a simple object oriented scripting language that is part of FlexWiki. It enables wiki articles to incorporate dynamic content and to customize the user interface. WikiTalk can access classes written in the .NET Framework and serves as an extension mechanism.

Federation and namespaces - one FlexWiki installation is associated with one federation that contains many wiki bases. A wiki base contains a namespace name and a list of other imported wiki bases. A link is constructed by specifying the namespace and the article name. If the namespace is not specified, the engine will first look in the current namespace, and then in all imported wiki bases. (FlexWiki: Wiki Federation Overview, 2005-11-12a)

Topic tips - when the mouse hovers over a link, a tool tip is shown with the summary of the page that the link points at. The tool tip also includes information about who and when last updated the page.

Page properties - metadata about an article is specified with a property name, a colon, and one or multiple values separated with comma. The metadata is for example used to define the summary used by the topic tip and to create a to do list. (FlexWiki: Wiki Federation Overview, 2005-11-12b)

Optimistic locking - if two users are updating an article at the same time, the user that finish the editing last, will get a message telling him that the page has changed while he was editing it.

(FlexWiki: FlexWiki Features, 2005-11-11)

3.3 MediaWiki

This is the wiki engine used by the Wikipedia project. It is written in PHP, and uses MySQL for storage. The software was originally built for the Wikipedia project, and contains a lot of functionality to handle the large number of users and the large number of different topic areas that the engine is used for. The code is licensed under the GPL license.

The current version is 1.5 that was released on October 5, 2005. The largest change was a modified database schema to enhance the performance and ease maintenance.

Since MediaWiki is used on the world largest wiki sites, it is probably the wiki engine that most people have used, and its set of features became a de facto standard. The huge traffic that MediaWiki can handle has impacts on its feature set, features that demand a lot of processing power are, in general, avoided.

MediaWiki has support for rendering complex math formulas using a combination of LaTeX, dvips, ImageMagick and Ghostscript (WikiMedia: Enable TeX, 2005-11-12).

The MediaWiki engine can run multiple instances on a single server, and use either the same database with different table prefixes or a specific database for each installation, but there is no support for running multiple wiki sites on the same instance (WikiMedia: MediaWiki FAQ, 2005-11-12).

The configuration of the wiki engine is achieved by modifying different configuration files and sometimes edits have to be made to the PHP source code. MediaWiki is just as FlexWiki a SourceForge project.

The following list of features is based on the MediaWiki documentation with comments and suggestions added by the author of this thesis. (MediaWiki - Documentation: Introduction, 2005-11-11)

3.4 Look and feel

A web site's success is not just about the content, the look and feel of a web site is a major factor for visitors' impression of both the quality of the information and the ease of use.

Most companies have a graphical profile, all their documents, advertisements and web sites should follow the recommendations of the graphical profile.

One CSS file should be used for the whole wiki, and all the articles should be presented using the same layout. Tags in the wiki markup language that enables the author to specify layout will soon create pages that do not share the same look and feel, and should thereby be avoided when possible.

3.4.1 Skins

A wiki engine should have at least one predefined look and feel that is usable out of the box, and that looks professional. A common feature is also to have layout themes, or as some programs and web sites call them, skins. A skin is a collection of images, CSS style sheets and for XSLT enabled applications, XSLT style sheets. The skin architecture should allow skin developers great flexibility in how the pages are rendered.

3.4.2 Readability

It is very common for web sites to use layouts that are not optimized for readability. Often an exciting design and maximum exposure of advertisement has been the main goal. This sometimes creates web pages that are both hard to read, and hard to get a good overview of.

The wiki markup should try to be as layout neutral as possible to enable skin developers great flexibility to create sites targeted at their user group.

One example of innovative thinking is International Herald Tribune's (www.ihf.com) user interface. They have a menu option called *Change format* where the user can chose between the one column layout, creating a long page where the user has to scroll, this is the layout used by the great majority of all web sites.

They also have a layout that shares the ideas from the newspaper edition, the text is in three columns, creating text lines that are very easy to follow and read. They have replaced the scrolling with two buttons, previous page and next page. These buttons are using DHTML to change the text, without reloading the whole page. It is also possible to click on the rightmost column to go to the next page, and on the leftmost column to go to the previous page.

On each page there is also possible to change the text size, this is saved for use on other pages on the site.

Another feature that could be useful on a wiki site is the clippings feature. Next to links to other articles and next to headings is a little symbol placed that looks like a document with a plus sign. Clicking this symbol adds that page to the Clippings menu. The content of the clipping menu seems to be stored as a cookie in the client's browser. If the user sees an article and wants to read it, but do not have time at the moment, then he adds that article to the clippings menu. On his next visit he can easily find it again. The functionally is similar to browsers' favorites or bookmarks feature, but this is implemented on the web site instead of in the browser.

3.4.3 Tool tips

The topic tips used by the FlexWiki engine are a great way to inform the user about an article without having to switch to that page. This is a feature that is also found in many Windows applications using the name tool tip.

3.4.4 Side bar

Wikimedia has support for a side bar, a box usually placed on the right side of the page to present a list of links. The side bar is often used in printed magazines and papers to give summary information, a list of items or some quote. The side bar is a way to grab the reader's attention and to make the most important information to stand out.

3.4.5 Stub threshold

A stub is an article with just the heading or just a few lines of text. It exists mostly to indicate that it would be a good idea to have an article containing information about a specific topic. Stubs can be detected by various methods, Wikimedia used the comma count method, an article missing a comma was

regarded as a stub but they no longer use this method since some languages do not use the comma symbol. A better method is probably to count the number of characters in the article.

Links to article stubs should be rendered different from links to full articles. This has two purposes, to indicate to a reader that there is no valuable content on the page that the links point at, and to indicate to an editor, that this is a page that needs attention.

Articles considered as stubs should also be noted on special stub list pages, where editors easily can get a look of all the articles that are not finished.

3.4.6 Printable version

There are multiple ways to create a page good for printing. It is possible to mark menu elements and other items that should not be part of the printout with CSS attributes that hides them when they are to be printed. A more common method is that there is a *printable version* link on each page, which links to the same article but rendered in a way optimized for printing.

The printed page should contain information about when the last edit time and date, the URL to the specific version of the article, and a link to the latest version of the article.

Often the license used for the content should be included in the printout, sometimes creating problems since the license is often longer than the article itself.

3.4.7 Report style

For some types of content, a more professional report style is needed. This could be archived with some template model, forcing the content of the article to follow a specific set of rules.

For scientific reports, this could add obligatory summary, introduction, discussion and conclusion sections. There should also be the option for long articles to include a table of contents. One thing that is not very often specified in wiki articles are references, which are very important for some subjects, and there should be a functionality to specify references in the same way as in a formal report, and to generate a list of references at the end of the article.

Numbered headings and other features used in formal report writing should if possible be supported.

Another feature that could be useful is to be able to group a set of articles together into a combined report or a book. This could be used to enable a user to print all pages related to a specific subject without having to print each individual article.

To take this one step further is to enable the content of multiple articles to be grouped together, and with the use of some functionality, like XSL-FO create a professional looking PDF of the set of articles. If this feature was combined with report style elements, it should be possible to write professional reports and thesis directly on the wiki. This would enable many authors to collaborate on one report, and give editors and reviewers functionality to provide instant feedback and progress reports.

3.5 Multimedia

A wiki engine should have the functionality for users to upload images, audio and video to the wiki. It should be encouraged to upload the media to the wiki instead of linking to it, this is to ensure that the content stays available. This type of content is often copyrighted, and the users that submit content must make sure that the media is licensed under terms compatible with the wiki's license rules.

There are text-to-speech engines that could be used to enable visitors to download an audio-file with the content of the page. This file could be used to increase accessibility and to enable users to listen to the content on their portable mp3-players, etc.

3.6 Version tracking

All the different versions of a wiki article should be saved in the archive. It should easily be possible to view or revert back to any previous version. When viewing an old article, it should be possible to choose if the links points to the latest version of their target articles, or the article that existed when the archived article was created.

This feature could be called snapshot view, for any point in time, it should be possible to browse the content of the wiki as it was at just that moment.

Older versions of media files, should, just as wiki articles, be preserved in the archive. There are two ways to link to media from an article, either the link is to one specific version of the media file, or the link is to the most current version, both types of links should be supported. When viewing an old archived article, it should be possible to decide what images are shown, either the images that existed at the same time as the article was created, or the most recent images.

3.6.1 Diff

A diffing feature is used to view what changed between two different versions, either just the changes could be shown, or the articles are viewed side by side, with changes highlighted. The functionality should be very similar to the diffing tools used by CVS and Subversion users.

3.6.2 Edit notification

To ensure the quality of the content it should be very easy to get notified when content is changed. This is to make sure that article edits are peer reviewed as soon as possible.

The notification should either contain the whole article, or just the sections that changed. There are many ways to deliver the notification, e-mail, newsgroup, IRC or other type of instant messaging system, RSS feeds etc.

A wiki engine should have a plugin system enabling easy extension of the notification features.

A user should be able to select what changes he should be notified about, this could include articles he has edited before, and their related articles, or any topic category, or something similar to FlexWiki's WikiNewsletter.

3.6.3 Backup

A wiki engine soon becomes a large repository of valuable content, it is very important that this content does not vanish in a system failure. All settings and all articles including archives and media files should be easy to backup. The backup should be easy and quick to restore, and there should also be backup options that provides more of export functionality, enabling content to be moved between different wikis.

3.7 Page types

The most common page type used is the article page type, this page type shows a specific version of an article, usually the latest version. A wiki engine needs to have support for other types of pages, the following should be a good start.

Archive - a dynamic page that shows a list of all versions of the article, containing information about modification time, the size of the edit, the name of editor and an optional edit remark.

Difference - a dynamic page showing the differences between the versions of an article.

Discussion - a wiki page that exists for each wiki article. Used by users to discuss the content of the article, and propose changes.

Voting - a dynamic page that can be used for voting. If editors disagree about the content of an article, a vote can be conducted, and hopefully the editors will follow the outcome of the vote. If not, there is the possibility of a wiki editing war, where two editors revert back to their version over and over again.

Who's who - a page that exist for each registered user, this page is only editable by that user and contains information about himself.

Special pages - dynamic pages that are needed for various functionality on the wiki web, such as user registration and administration.

3.8 Editing

There is a large variety of different markup languages for wiki content, most are some type of dialect of wiki text. This is a problem for editors who contribute with content to wikis using different dialects since they have to know several dialects and not mix them together.

The editing should, if possible, be independent of the storage format. This can be achieved with the help of a plugin system that translates the edited content into the markup language used internally.

For some type of articles, it should be possible to use templates that either suggest a certain format, or requires a certain format. This can be used to enhance the common look and feel, and also the quality of the content. One example is book reviews, if an editor wants to write a book review, he can choose a book review template, and then certain information that is mandatory, like author's name, title and ISBN should be present.

3.8.1 Smart tags

There are certain types of information that needs to be dynamic, the front page might want to list the total number of articles in the wiki. This functionality could be constructed with the help of special tags that are replaced at request time with content.

3.8.2 Multilanguage support

The internet has truly become global, but still there is often limited support for non-western languages. The content should be saved as UNICODE preferably using 16 bits per character. The markup language should support right to left languages, and other features required for true multilingual support.

3.9 Search and queries

The articles should be searchable both on article name and meta tags, and on the whole content of the article. For the whole article search it is possible to use an external search engine like Google, but preferably there should be an onsite free text search function.

The free text search and the article name search have to use indexes to speed up searching. Using regular expressions or other methods to search each article does not scale, and even if it was used by many early wikis it is not an option today for anything than very small wikis.

3.10 Spam and vandalism protection

Spam and vandalism can be a major problem for some wikis. There are multiple type of spam, and different types of vandalism. A wiki engine should have features to minimize the amount of spam and vandalism. The notification features of a wiki are very useful here. The better the notification features are, the quicker an editor can revert the article to an older version without the spam or vandalism.

3.11 Extensibility

It is probably not possible to create a wiki engine that incorporates features for every possible usage scenario, instead the wiki engine should have good support for plugins that can enhance or replace the functionality in the wiki.

Chapter 4

Document formats

Even thus a wiki could contain text without any markup, even the first wiki engines supported a limited set of markup tags. Markup can be divided into two categories, structural markup and layout markup. Structural markup defines the structure of a document, such as headings and paragraphs. Layout markup defines how the content should look, for example that a specific paragraph should be italic. The markup tags can look very different from one document format to another which becomes a problem when content should be moved from one system to another.

This chapter deals with different types of wiki markup, the current situation and methods that are used. It defines a set of requirements that a new wiki markup language should be able to handle and discusses existing wiki and none wiki markups to get a good foundation to use when formulating a new XML wiki markup language.

4.1 Current situation

Today, most wikis use a markup language called wiki text. It was designed to be written by hand and to be edited inside an HTML textarea element. The markup is simple to write since it is made up of normal characters, but it has evolved to handle complex documents containing tables and text formatting and has lost much of its original simplicity.

As the wiki concept became more and more popular, different wiki engines started to emerge and the wiki text tags were extended with new ones to support more advanced features. There was no overall standardization, which resulted in a large set of incompatible wiki text dialects. This is an obstacle for wiki contributors that contribute with content to different wikis using different dialects, since they have to know every specific dialect to be able to write in such markup.

4.1.1 Different dialects of wiki text

The wiki text dialects are not 100% compatible with each other, even worse, different dialects uses the same markup elements for different things. This is a well known problem, and there are many efforts underway to find a solution to

this. The large number of existing documents should ideally be converted into a new standardized wiki text language in a more or less automatic way.

Instead of using one common wiki text language, there are proposals to at least define a common interchange language that every wiki should be able to import and export from.

The fact that more and more wikis accept a subset of HTML tags and even CSS attributes will put a lot of demands on a common wiki text language that is a superset of all existing wiki text dialects.

A superset language is not an optimal solution to the problem, since it would create an even more bloated wiki text standard. Instead wiki text should look back to its origins. The first wiki engine implementation, done by Ward Cunningham in 1995, was described as

the simplest online database that could possibly work (Leuf and Cunningham, 2001)

There have been a lot of improvements in the programming tools and in the browsers since 1995. There is a need to ask the question: What is the simplest online database that could possibly work today?

There are some parts of existing wikis that are not done in the simplest possible way with the technology and standards that exist today.

Wiki text - a proprietary, non-standard markup language with lack of, or very relaxed, validation. The limited validation can create ambiguous output, and the resulting (X)HTML code can be non conformant to the (X)HTML standards.

Transform language - wiki text have no standardized transform language into HTML or any other format. Most wiki engines use a general purpose scripting language for the transformations.

Text area editing - this is still the only option for browsers on platforms with limited resources, like on a mobile phone or a PDA. On desktop computers, the most popular browsers are Mozilla Firefox and Internet Explorer. Both these browsers have support for rich text editing, which gives the user a what you see is what you get (WYSIWYG) experience, or at least, what you see is nearly what you get. This is not to imply that text area editing does not have its benefits over rich text editing, but they are optimal for each set of user groups. Just as an HTML Editor is good for some users, and Notepad is good for some HTML gurus.

Object orientation - most wiki engines are written in scripting languages that are loosely typed and the implementations are often not object oriented, making them hard to extend and maintain.

Configuration - usually the configuration of a wiki engine is a mix of modifying configuration files and often layout changes has to be made in the scripting code.

4.1.2 HTML and XHTML

There are some wiki engines that allow the user to directly specify (X)HTML (HTML or XHTML) markup. This can be as an extension to wiki text, for ex-

ample everything wrapped inside a starting and ending `<html>` tag is interpreted as (X)HTML markup, or the (X)HTML can be the only markup language.

The power of (X)HTML is the large number of elements and in combination with CSS it is possible to specify almost any layout. This flexibility has its price, the documents will very easily not share a common look and feel, since it is so easy for an editor of an article to specify their style preference. The markup will probably shift its focus from structure markup into layout markup.

The mix of (X)HTML and wiki text can create markup that is quite complex, see the markup example from part of Wikipedia's English main page. Notice the mix of wiki text markup, html elements, attributes and CSS style properties.

```
<div style="padding-bottom: .3em; margin: 0 .5em .5em">
  {{Main Page banner}}
</div>
{| cellspacing="3" |- valign="top" |width="55%"
  class="MainPageBG" style="border: 1px solid #ffc9c9;
  color: #000; background-color: #fff3f3"|
<div style="padding: .4em .9em .9em">
===Today's featured article===
{{Wikipedia:Today's featured article/
{{CURRENTMONTHNAME}} {{CURRENTDAY}}, {{CURRENTYEAR}}}}
```

(Wikipedia: Main page, 2005-11-13)

There are major security concerns with allowing the user to specify (X)HTML markup. Filters that remove potential harmful code have to be implemented. Examples of harmful code are blocks or attributes with JavaScript, that either exploits some browser bug to get access to a user's local file system or redirect the user to some other page containing advertisement etc. The (X)HTML could also contain object elements, asking the user to install some plugin that could contain a virus, spyware or adware.

The public wikis that allow (X)HTML must have this type of filter, and there might be many none obvious way of embedding harmful code into a page.

One way of limiting the above problems with lack of common look and feel and security issues is to only enable a subset of (X)HTML elements and attributes. If the wiki is using XHTML this could be accomplished with a relatively simple XSL-stylessheet that strips disallowed elements and attributes away from the document. If the wiki is using HTML, then it is not possible to use XSL, instead there need to be a custom implemented parsing filter.

XML wikis

No commonly used wiki implementation is using XML to both edit and view the pages. The Open Wiki engine supports traditional wiki text editing, but translates the results and saves it internally as XML. The pages are then transformed on the fly with XSLT, either server side or client side. The implementation is written in 100% classic ASP. (Open Wiki: Xml Formatting, 2005-11-13)

4.2 Markup Requirements

The first step in defining a new markup language is to identify the requirements. The requirements will probably vary greatly between different types of wiki

usage. Some overall requirements are defined in this section.

Well formed and valid - an edited article should be parsed to make sure that the markup is well formed and valid and the errors should be clearly reported to the user. This is often not ensured in wiki text implementations.

Rich presentation - the markup should include a minimal set of tags to specify layout, and the focus should be on tags specifying structure. There should be tags specifying structure that enable the document to be presented in a visual interesting way.

Easy editing - the language should be easy to read, both for a human and for the computer. The markup should both be possible to edit by hand and with the help of an editor.

Extensible - wiki sites contains a large variety of different topics, and it should be easy to add extra tags to extend the functionality. If possible, the tags should be added in such a way that a wiki without the extension should still be able to parse and display the content in a default way.

Media neutral - today, most wiki sites are read from the screen, but as discussed in other chapters it would sometimes be good to enable flexible high quality output to paper, and with the increased use of mp3-players and better text-to-speech engines, wiki content should also be possible to be transformed to audio files.

Standardized - the markup should become a standard, used by many different wiki engines. The markup should be easy to transform into another markup language.

Fast presentation - the markup should enable quick transformation into (X)HTML.

Internalization - UTF-16 should be used, to support as many languages as possible. The markup should be able to express structure in any language.

4.3 Existing popular markup languages

There are many ways to markup data, most of them are proprietary, and used by only one or a few applications. The trend is to use markup that specify structure instead of layout, here is a quick review of some of them.

4.3.1 LaTeX

This thesis is written in \LaTeX , it is a format designed for manual editing, to be used to write articles, reports, books and letters. It has extensive support for math formulas.

A report for example, is divided into chapter, sections, subsection and sub-subsections. There are tags for specifying lists, quotations, different font sizes etc. LaTeX can easily be extended by loading additional packages and existing tags can be redefined. The LaTeX source is often transformed to postscript or PDF.

One useful feature is the ability to easily write comments in the document, that are not visible when transformed to postscript or PDF.

An example of LaTeX markup:

```
\documentclass[a4paper]{report}
\usepackage[latin1]{inputenc}
\title{Collaborative web content management\ - Wiki and beyond}
\author{Mattias Beermann}
\date{\today}
\begin{document}
\maketitle

\chapter{Introduction}
The basic idea behind the wiki concept is simple...
\section{History}
...

\end{document}
```

4.3.2 DocBook

DocBook is an XML markup language designed to be used for technical documentation. It started as an SGML application, but today most documents are written using XML.

It is standardized and maintained by the DocBook Technical Committee at OASIS.

Many open source projects use DocBook for their documentation, some of them are the Linux Documentation Project, the Linux Kernel and GNOME.

There are DSSSL and XSL stylesheets to transform DocBook files to HTML, PDF, RTF and many other formats.

The XML format can be edited in a normal text editor, but there are many editors, both free and commercial, that have support for DocBook. (Wikipedia: DocBook, 2005-11-14)

An example of DocBook markup:

```
<book id="cwcm">
  <title>Collaborative web content management</title>
  <chapter id="chap1">
    <title>Introduction</title>
    <para>The basic idea behind the wiki concept is simple...</para>
    <sect1 id="sec-history">
      <title>History</title>
      <para>...</para>
    </sect1>
  </chapter>
</book>
```

4.3.3 (X)HTML

HTML is the markup language used to specify web pages. The first draft was published in June 1993. HTML is an acronym for Hypertext markup language,

and it was HTML that made internet what it is today.

The language has very relaxed rules, especially the first versions of HTML were very forgiving for syntax errors. This led to problems since different browsers handled the syntax errors in different ways. Each browser vendor added new features to the language that just their browser could support.

The language was standardized by W3C, which made the language stricter and added support for new functionality. HTML 3.2 was published in January 14, 1997, HTML 4.0 in December 18, 1997 and HTML 4.01 in December 24, 1999. On May 15, 2000 ISO HTML based on HTML 4.01 Strict became an ISO/IEC international standard.

The HTML standard used a mix of elements and attributes specifying structure and layout. This led to the development of CSS, that was designed to specify layout separated from the HTML markup. (Wikipedia: HTML, 2005-11-14)

The introduction of XML, led to an XHTML 1.0 standard that was conformant to XML rules. XHTML was a great step towards wider adoption of CSS since many of the layout elements were deprecated.

The Extensible Hypertext Markup Language (XHTML) is a family of current and future document types and modules that reproduce, subset, and extend HTML, reformulated in XML. XHTML Family document types are all XML-based, and ultimately are designed to work in conjunction with XML-based user agents. (W3C: HTML Home Page, 2005-11-14)

Currently the XHTML 2.0 specification is in progress, with the seventh public draft published on 27 May 2005. The XHTML 2.0 specification is much stricter than any previous version and includes a smaller set of elements and is not backward compatible as its earlier versions. (W3C: HTML Home Page, 2005-11-14)

An example of HTML markup:

```
<html>
  <head><title>Collaborative web content management</title></head>
  <body>
    <h1>Introduction</h1>
    <p>The basic idea behind the wiki concept is simple...</p>
    <h2>History<h2>
    <p>...</p>
  </body>
</html>
```

An example of XHTML 2.0 markup:

```
<html>
  <head><title>Collaborative web content management</title></head>
  <body>
    <h>Introduction</h>
    <p>The basic idea behind the wiki concept is simple...</p>
    <section>
      <h>History</h>
```

```
    <p>...</p>
  <section>
</body>
</html>
```

4.3.4 XML

The XML markup language is a general purpose markup language with its roots in the SGML-standard. XML is an abbreviation for Extensible Markup Language. It is a W3C recommendation as of 1998.

The design goals for XML as specified by W3C were:

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.

(W3C: Extensible Markup Language 1.0, 2005-11-14)

XML is basically a set of rules of how to define a specific markup language. It states that all elements should begin with a < and end with a >. An element must have a name, and may have one or several attributes. Elements must be properly nested, they can not overlap. The syntax looks very similar to HTML. It is up to the developer to specify what element names and attributes to use, this can be formalized into a document type definition (DTD), XML Schema or some other schema language.

An XML document is considered well formed if it follows the rules of the XML syntax and if the document also follows the rules specified in the related DTD and XML Schema, it is considered valid.

One large advantage with the XML format is that the parser must not read a document that is not well formed, avoiding different interpretations between implementations as have been a huge problem with HTML.

XML has quickly become a very successful standard that is widely used to store information and to exchange information between applications. A large group of related standards have contributed to the success of XML.

There are parsers available for XML content in most languages, and they often implement the XML Document object model, a W3C specification that specifies an API to parse and modify XML documents. There are also other APIs used to process XML documents.

An example of an XML document:

```
<document>
  <heading>Introduction</heading>
  <paragraph>The basic idea behind the wiki concept is simple...</paragraph>
  <section>
    <heading>History</heading>
    <paragraph>...</paragraph>
  </section>
</document>
```

The rest of this chapter discusses how XML can be used to express wiki content.

4.4 WXML - a wiki XML markup language

Each markup language has its design goals and uses. To use an existing markup language to markup wiki content is possible, both L^AT_EX, DocBook and (X)HTML should work, but not in an optimal way since they were not designed for wiki content.

There is no reason to invent everything again, instead the best ideas from other markup languages should be used and combined into a markup language for wiki content using XML markup.

The language should have support for extensibility and should try to ease the transition from wiki text to the wiki xml markup language (WXML).

The language should be simple to use and offer a stable and lightweight base that different wiki engines can extend upon.

The wiki XML markup language designed in this thesis is an experimental language that can be used to test a wiki implementation using XML for editing, storage and presentation. To design a language that can be replace wiki text and become a standard requires cooperation between the developers of the largest wiki engines and from the wiki community. This chapter provides a prototype that only includes the very basics.

The following sections will use a simplified but similar definition to the one used in the XHTML 2.0 recommendation. (W3C: XHTML 2.0 Draft, 2005-11-15)

4.4.1 Style

Different XML vocabularies use different styles, this is mostly a matter of personal preference but it has some other implications. The element names can be all uppercase letters (<LAYOUTMASTERSET>), which is uncommon. A more common approach is to use mixed casing, either Pascal casing where all words starts with an uppercase letter (<LayoutMasterSet>), or camel casing, where all words starts with an uppercase letter except the first word (<layoutMasterSet>). The XSL family of standards all use lowercase letters, words are separated with a dash (<layout-master-set>). Pascal and camel casing are often used in programming languages' coding conventions, but for this XML vocabulary the XSL style is used, due to the good readability.

The elements and attributes will be in English, implementations specifically targeted at some other language should still use English element and attribute names, but might expose translated element and attribute names when the document is manually edited.

Some standards, like XHTML and DocBook use abbreviated names, the names becomes faster to manually write, and requires less storage space and faster retrieval, but considering that most structure based markups have a majority of text compared to element and attribute data, this is not a major benefit. If an XML vocabulary uses abbreviations there has to be some rule specifying how the abbreviations should be constructed, DocBook uses `<para>` and XHTML `<p>`, and these standards only abbreviate some names and not all. One of the design goals of XML was: *Terseness in XML markup is of minimal importance*. The WXML will use no abbreviations, instead the element and attribute names will be written in full, this is also in line with the XSL family of standards. This has the benefit that an editor writing the XML markup manually always know it is the full name that is used and does not have to think about if it is p, or para, or some other abbreviated form.

4.4.2 Modularization of XHTML

XHTML shares some of the same problems that a wiki markup language will face. The XHTML standard has been carefully designed and developed, and solutions and ideas should be copied from that standard whenever possible. This has two advantages, there is no need to reinvent solutions to the same set of problems and developers that know XHTML will quickly understand the WXML language.

The problem XHTML faces is that some devices does not implement support for the whole XHTML standard, instead they only implement a subset of it, and some devices needs extra functionality not provided in the XHTML standard. The same problems exist with a general purpose wiki language and the same solution should be possible to use, in XHTML it is called Modularization of XHTML.

Modularizing XHTML provides a means for product designers to specify which elements are supported by a device using standard building blocks and standard methods for specifying which building blocks are used. These modules serve as *points of conformance* for the content community. The content community can now target the installed base that supports a certain collection of modules, rather than worry about the installed base that supports this or that permutation of XHTML elements. (W3C: Modularization of XHTML 1.0 - Second Edition (Working draft), 2005-11-14)

The XML elements and attributes are grouped into modules. A certain implementation can specify which modules it supports. This is a simple and effective way of creating a modular and extensible design. The interested reader can read the whole specification for more information.

4.4.3 Metadata and article markup

There are two distinct parts, one is the wiki article markup, the content of the article, and then there is the metadata about the article. The metadata can contain information about editors, when the article was edited, the license used and other properties.

4.4.4 Core modules

To preserve one of the key principles with the wiki concept, *less is more*, the required core modules should be kept to a minimum, to enable small and simple implementations. The modules are loosely defined in the same way as XHTML 2.0. (W3C: XHTML 2.0 Draft, 2005-11-15)

Document module

| Elements | Content Model |
|----------|-------------------------|
| wxml | metadata, article |
| metadata | title |
| title | PCDATA* |
| article | (Heading Structural)* |

Structural module

| Elements | Content Model |
|-----------|--------------------------|
| heading | PCDATA* |
| section | (PCDATA Flow)* |
| paragraph | (PCDATA Text List)* |

The content model for this module defines some content sets:

Heading: heading

Structural: List | paragraph | section

Flow: Heading | Structural | Text

Text module

| Elements | Attributes | Content Model |
|-----------|------------|------------------|
| emphasize | | (PCDATA Text)* |
| link | href | (PCDATA Text)* |
| quote | | (PCDATA Text)* |

The content model for this module defines a content set:

Text emphasize | link | quote

List module

| Elements | Content Model |
|-----------------|-------------------|
| unordered-list | item+ |
| ordered-list | item+ |
| item | (PCDATA List)* |
| definition-list | entry+ |
| entry | term, definition+ |
| term | PCDATA* |
| defintion | (PCDATA List)* |

The content model for this module defines a content set:

List unordered-list | ordered-list | defintion-list

Wiki text module

To be able to slowly phase in WXML, it should be possible to specify wiki text markup inside a WXML document. A module designed for backward compatibility could be used. To convert existing wiki text markup to WXML, the markup is automatically transformed into WXML, tags that has no correspondent element in WXML should use elements from the Wiki text module.

When a document is edited by hand, the editor should be notified that the article contains elements that are deprecated and that they should be removed if possible.

Links

The link functionality provided in WXML is so far very limited. The current link element is just to provide basic functionality until something better is added. The XLink standard contains a lot of information about different types of links. The XLink recommendation has existed since 17 June 2001 but has so far not been widely implemented. It is not sure that XLink provides the functionality needed for wiki linking needs, but some of the ideas presented in that standard could be useful. (W3C: XML Linking Language 1.0, 2005-11-15)

4.4.5 Example of a WXML document

This is a simple example of a WXML document:

```
<wxml>
  <metadata>
    <title>Collaborative web content management</title>
  </metadata>
  <article>
    <heading>Introduction</heading>
    <paragraph>The basic idea behind the wiki concept is simple...</paragraph>
    <section>
      <heading>History</heading>
      <paragrpah>...</paragraph>
    </section>
  </article>
</wxml>
```

4.5 Transformation of XML documents

XML is very easy to transform into other representations, either by writing code that uses XML DOM or some other parser to read one document and output another one, or to use the XSL stylesheet language.

The extensible stylesheet language (XSL) family consists of three parts. They are the extensible stylesheet language transformations (XSLT), XML Path Language (XPath) and XSL formatting objects (XSL-FO).

XSLT is used to describe a set of transformations that converts one XML vocabulary into another XML vocabulary. It is also possible to transform XML documents into HTML and text documents.

XPath is used to address and access various parts of an XML document. It is used in XSLT to specify the XML elements and nodes that should be accessed.

XSL-FO is an XML vocabulary used for describing formatting semantics. With XSLT an XML document can be transformed into the XSL-FO vocabulary and then for example turned into PDF. (W3C: The Extensible Stylesheet Language Family, 2005-11-14)

4.5.1 Overview of XSLT

XSLT is one of the standards that have made XML so popular. XSLT defines an XML vocabulary that is a very powerful tool to describe both simple and complex transformations between different XML vocabularies. The syntax can at first glance look very verbose and even complicated but it is easy to learn the basics of it, and it quickly becomes a very handy tool for transforming XML documents.

Input XML document:

```
<document>
  <title>Foo</title>
  <paragraph>Baz</paragraph>
  <paragraph>Bar</paragraph>
</document>
```

The above document should be transformed to a HTML document that should look like:

```
<html>
  <head><title>Foo</title></head>
  <body>
    <h1>Foo</h1>
    <p>Baz</p>
    <p>Bar</p>
  </body>
</html>
```

This could be accomplished by writing a small program that reads the source XML document into an XML DOM object, and then some logic that outputs the resulting HTML document. A SAX parser could be used to read the source

document, and event listeners added to the sax parser contains logic to output the resulting document or the following XSLT-stylesheet could be used.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match='/document'>
  <html>
    <head><title><xsl:value-of select='title'/></title></head>
  <body>
    <h1><xsl:value-of select='title'/></h1>
    <xsl:apply-templates select='paragraph'"/>
  </body>
</html>
</xsl:template>

<xsl:template match="'paragraph'">
<p><xsl:value-of select="'"'/></p>
</xsl:template>

</xsl>
```

This is an example of a very simple XSLT-stylesheet, using just two templates, it is possible to have many templates, for loops, if statements and other types of logic. The syntax of XSLT enables a good XSLT developer to quickly write advanced transformations.

An XSLT stylehseet can contain include statements to other XSLT stylesheets, enabling a flexible way to reuse existing stylesheets.

Chapter 5

Wiki article editing

Most wiki engines use a HTML textarea element to display the wiki text. There is sometimes a toolbox with buttons for inserting code for various frequent tags. This type of editor requires only standard HTML and works in almost any browser. The toolbox is optional, it requires basic JavaScript functionality.

To enable editing of long articles, wiki engines often provides the option to either edit the whole article or just a section of it. The section choice makes it much easier to just change a few lines.

Some wiki text dialects that allow a mix of HTML, CSS and wiki text enables an author to write very confusing code that requires HTML knowledge, CSS knowledge and wiki text knowledge. This creates a large barrier for people without this knowledge. It should be noted that most articles are only using a few basic wiki text commands. This basic set of commands are able to express most of the structure needed in most articles.

There are some wiki engines using WYSIWYG editing, often these wikis use HTML instead of wiki text as the storage language.

There are three major solutions to enable editing of a wiki articles using XML markup:

- Textarea XML editing
- Wiki text editing and conversion to XML before the article is saved.
- WYSIWYG XML editing

5.1 Textarea XML Editing

A wiki article written using XML markup can be edited in the same way as wiki text is edited, in an HTML textarea element. This works in all browsers supporting the HTML textarea element.

A toolbox can be provided that inserts the different elements.

XML needs to be well formed, all tags have to be closed, and the nesting of elements needs to be correct. This is something the author has to be aware of and make sure to follow. There are also some characters that need escaping, the most common is the & symbol, which has to be specified as &.

The XML document should also be well formed, that is, it should conform to an XML DTD or schema, this is another problem with manual editing.

The well formness check and validity check can be made at the client by loading the content of the textarea into the browser's XML DOM implementation, and show the parsing errors to the user. But the majority of error messages will be too technical for most users, and even if the implementation will be very simple, it will not give a good editing experience for authors that are not aware of the XML standard.

An experimental wiki engine could use this type of editing to begin with, since it is implementable in just a few lines of code.

5.2 Wiki text editing

The wiki article specified in XML could be transformed with XSLT into wiki text. The wiki text could be edited, using the HTML textarea method. Since the wiki text has limited well formness constrains and validity constrains, it is easier for the author to write acceptable wiki text markup. The large numbers of existing contributors to wiki sites using wiki text are accustomed to this method, and to them the underlying storage format should not matter.

The problem is to parse the wiki text back to XML and enforce the stricter rules. This parser has to be able to fix common wiki text well formness problems, and for more complex cases, give feedback to the author, to inform him or her where the problem is.

5.3 WYSIWYG XML editing

The most usable type of editing for the majority of users is probably one based on a WYSIWYG editing. The wiki article is displayed in the same way, or in a similar way, as when the article is read. The author can directly edit the text and use toolbars to format the text and get instant feedback of how it will look. This is very similar to how most word processing programs works.

For a more experienced author, there can be the option to switch to source code view where he can view and edit the XML code directly.

5.4 Experimental implementation

One part of this thesis was to create an experimental WYSIWYG editor for editing XML data in Internet Explorer 6.0 and Mozilla Firefox 1.0.

5.4.1 HTML WYSIWYG editors

There are many HTML WYSIWYG editors, some of the uses are for content management systems, e-mail composing and blog writing. There are two types, the most common type uses the HTML editing features provided by the browser, and adds a graphical user interface in the form of toolboxes and menus. Usually an image upload feature is also provided.

The other type is a component, often in Flash, Java or ActiveX, that provides the HTML editing features. This method has the benefits of having a richer programming language to use, and that the developer can specify exactly how the editor should work. The drawback is that it is often quite a lot of code that

is required, and that the user might have to install the component and allow it to run locally on their computer. (htmlArea, 2005-11-15)

5.4.2 Different approaches

The experimental implementation was written using the editing functionality available using JavaScript and DHTML. There are three different approaches:

An editor from the ground up - the editor could be written from the ground up using JavaScript and DHTML without using the editing features provided by the browser. This is the most flexible solution, but also the one that probably requires the most lines of code. The lack of features in the browser would probably make some operations like, cut, copy, and paste hard or impossible. The text cursor has to be faked, using an image, text navigation would probably also be very hard to implement, there are a lot of commands that a user expects to work, page up, page down, ctrl+left, ctrl+right, all would have to be implemented in JavaScript. It is very doubtful if it is possible to write a full featured implementation in this way due to the limit set of features and properties exposed by the browsers.

contentEditable = true This is not supported yet by Mozilla Firefox, but it works in Internet Explorer 6.0. This property can be set on almost any element of an HTML page, making the content editable and enabling a text cursor. To add HTML elements, these have to either be pasted, or inserted with the help of JavaScript code.

designMode = 'on' This property is set on document level, and enables the whole document to be edited. To limit the editable portion, an IFRAME is often used to wrap the editable document. The document containing the IFRAME provides the logic for menus and toolbars.

The implementation use contentEditable for Internet Explorer, since it provided some better functionality, and designMode in Mozilla Firefox.

5.4.3 Lack of XML support

The designMode editing allows editing of HTML content, not of XML content. Ideally the browsers should support editing of XML documents, and enforce the DTD/Schema of the document. The XML document could be formatted with CSS to look similar to the final page.

Since this is not the case, a workaround had to be found. The solution used was to transform the XML document into XHTML, and using the class attribute to specify the original XML element name.

To transform the document back to WXML, the edited document is parsed, and the WXML elements are created based on the class attributes.

5.4.4 Cut and paste

Cut and paste functionality is a *must have* in an editor. An author must be able to move content from one place in an article to another. The copied text should preserve the original formatting when possible.

Mozilla Firefox has no on paste event. It took a lot of effort and time to work around this issue, by carefully monitoring the text cursor, it is possible to detect a paste, since the text cursor position will change. The pasted content has to be selected, also a hard task using the functions provided by Mozilla, and the pasted content could contain HTML code from a normal HTML page, containing tags that are not allowed in the WXML. These tags must be removed.

If some text is selected, and then the user paste something, there needs to be logic that makes sure the resulting document is a well formed and valid WXML document.

5.4.5 DTD/Schema awareness

Most of the editing features are provided by the browser, but since they are not XML aware there needs to be a lot of code making sure that the document stays valid at all time, no matter what the user does.

With the limited set of features provided by the browser, this is not an easy task.

The user should only be able to insert valid elements at the current position, thus the toolbox needs to be updated based on the current position in the document.

The code that provides this functionality, and other validation dependent code, can be written in two ways, either it is written for one set of WXML modules, or it is written in a more general way that reads a DTD or Schema and based on that allows or disallows certain actions. It is probably very hard to make it fully schema aware, due to the large set of features found in the Schema language, instead a simplified schema that is sufficient for WXML modules could be used.

5.4.6 Browser issues

The limited set of features provided made the developing experience very slow and hard. The implementation was written to work in both Mozilla Firefox and Internet Explorer. They share some of the same objects, but very often the code has to take different paths dependent on the browser. This is one example of the different code needed to support both browsers:

```
var kxml;
if (window.DOMParser) {
    var parser = new DOMParser();
    kxml = parser.parseFromString("<content>" + this.textarea.value +
        "</content>", "text/xml");
}
else if (window.ActiveXObject) {
    kxml = new ActiveXObject("Microsoft.XMLDOM");
    kxml.async = false;
    kxml.loadXML("<content>" + this.textarea.value + "</content>");
}
```

The window.ActiveX function is provided by Internet Explorer and window.DOMParser by Mozilla Firefox. The above example demonstrates the correct way to detect browser differences. The correct way is to test if the feature

exists, if it does, then it is used. Some developers write code that get the type and version of the browser, and use that to decide which code to run. Both ways work but if for example Safari implements a feature in the same way as Mozilla, then it will just work with the above code but if the browser type and version statement had been used, the code has to be modified to also add the Safari case.

The development quickly becomes trial and error, since most functions are poorly documented, and cross browser issues are common, it is very common to write one or a few lines, test, try something else, test again, and repeat until it works.

The implementation that was written to evaluate the XML editing possibilities was written in around 500 lines of JavaScript code. The implementation showed that there seems to be ways to work around most issues, and that copy and paste can be made to work in both Internet Explorer and Mozilla Firefox but with a lot of complicated code. This feature was not found to be implemented in any of the HTML editors reviewed before the implementation was done, so that was considered a success for this implementation.

The DTD/Schema awareness were never implemented, due to time constraints, but this part should not be limited by lack of browser features, since it is mostly about writing JavaScript code that takes care of all the logic.

5.4.7 Future work

The implementation showed that it is probably possible to write a pretty good WYSIWYG wiki XML editor using JavaScript that has support for both Mozilla Firefox and Internet Explorer. The implementation also showed that this requires much more time and work than originally thought.

Future work should focus on DTD/Schema awareness, and to provide an editing experience that is as similar as possible compared to modern word processors. The problem of ensuring that the edited document stays valid during the editing can not be underestimated.

5.5 Multiple required editors

The textarea XML editing is very simple to implement, and would probably be the first editing mode that an XML wiki engine should implement. It enables developers, and contributors, to write and edit articles. The lack of usability is made up by the easy implementation.

The wiki text editing is important to make the large number of existing contributors that know wiki text to feel comfortable. One important feature to make an XML wiki popular is to enable automatic import of content from wiki text, thus there needs to be an import module that can convert wiki text to XML. This import module could easily be modified to also support wiki text editing and then transform the result back to XML.

The WYSIWYG XML editing feature is the best alternative, it would give existing wiki text contributors a new better editing experience, and at the same time it would make it easier for new contributors to edit and create new articles.

A wiki engine evolves, and should have a plugin system, enabling different editing modes. The textarea XML editing mode should be used for hardcore

users, and to be used before any other mode is available. The wiki text editing enables the wiki editing experience to be similar to existing wiki engines, making a transition easier. The WYSIWYG XML editing is a feature that should be implemented when it is possible to create such functionality in an efficient way.

Chapter 6

Architecture and design considerations

The first wiki engines were simple hacks that were designed to be as simple as possible. The focus was on a rich enough feature set, performance was not a major concern. This was sufficient then, but today one of the largest and most popular web sites on the internet, Wikipedia (wikipedia.org), ranked number 38, is a wiki running on the MediaWiki engine (Alexa - Global Top 500, 2005-11-05). Wikipedia has large problems to keep up with the ever increasing traffic and data growth.

This chapter discusses best practices for creating a software architecture and design that has both high performance and good scalability. For more information the interested reader can read the book *Improving .NET Application Performance and Scalability* (Meier, Vasireddy, Babbar, and Mackman, 2004) for a vast amount of information related to improving .NET framework performance and scalability.

There are multiple ways to architect and design both client and server side applications. For client applications performance is often not a problem since the application is run by one user on a dedicated machine. The opposite is true for a server application, which has to handle many simultaneous users, creating a large load on the server machine.

There is always a tradeoff between designing for performance and scalability compared to designing for simplicity and manageability. Depending of the expected use of the application, it is not always motivated to design for maximum performance since the complexity increases and thereby the development cost. If the application is scalable, it can be better to spend money on increased server hardware performance or on multiple servers than to optimize the code.

This chapter will focus on the requirements on implementing a wiki engine with high performance and good scalability and at the same time trying to keep the implementation simple and easy to manage and understand.

Performance and scalability is not something that can be put on at the end of a project, it has to be taken into account during the whole software development lifecycle.

If you're very lucky, performance problems can be fixed after the fact. But, as often as not, it will take a great deal of effort to get

your code to where it needs to be for acceptable performance. This is a very bad trap to fall into. At its worst, you'll be faced with a memorable and sometimes job-ending quote: 'This will never work. You're going to have to start all over.' - Rico Mariani, Architect, Microsoft

6.1 Performance objectives

The expected workload on a wiki engine running a large wiki site is too large to enable a single server solution, instead the focus should from the beginning be to achieve as good performance as possible, and to enable the application to scale out to multiple servers.

6.1.1 Wikipedia.org as a performance example

Wikipedia is running on the MediaWiki server software, a wiki engine written specifically for Wikipedia. The implementation is written in PHP, using MySQL as database backend. Wikipedia runs MediaWiki on Apache web servers running Linux. This is usually referred to as a LAMP setup, Linux, Apache, MySQL and PHP.

In February 2003, the database was roughly 4 GB, excluding images and multimedia, in August 2003 around 16 GB, this continued to grow and in April 2004 the database had grown to 57 GB with a growth rate at above 1 GB a week. In October 2004, the database had reached 170 GB. (Wikipedia: Technical FAQ, 2005-11-05)

All texts on Wikipedia are licensed under the GNU Free Documentation License, and thereby it's free to use under the condition stated in the license. Complete database dumps are available for download, and there are some web sites that bring Wikipedia content into their own site, for example answers.com and yahoo.com.

Hardware

The hardware used to run Wikipedia consist of roughly 89 machines in Florida, 5 near Paris, 11 in Amsterdam and 23 provided by Yahoo. In 2004, they got 39 new servers, and in 2005 88 new servers. The machines can be divided into four categories, database servers, web servers, Squid servers and others. (Wikimedia servers, 2005-11-05)

The main database server stores all articles in a MySQL database. The disks are RAID 10 and RAID 1, with battery backed up cache. This is to ensure that no data is lost in case of disk failure or power failure. The server has 8 GB of RAM, 5.8 GB of these are used for database caching, giving a hit rate of over 99%. The high hit rate is probably due to the fact that most rows in the database contain old pages, which are not as frequently accessed. There are also slave database servers.

The web servers are running MediaWiki on Apache, using Turck MMCache to speed up PHP execution. All web servers have the same software configuration.

The Squid servers run the Squid proxy and web cacher server software. Frequently requested pages will reside in the Squid servers' cache, and thus they

can serve the request directly and do not have to access the web server. They have a cache hit rate of around 78% percent, thereby greatly offloading the Apache servers. This is particularly useful when there is a large share of request to a specific page. For example, say that some large web site, such as a popular newspaper has a link to a specific page on the wiki, then the Squids can take care of all those requests. The Squid servers are configured to do load balancing on the Apache servers, the request to the Squid servers are balanced by round robin DNS. Wikipedia started to use Squid servers in February 2004.

The other category, are machines that handles general purpose tasks, such as mail, DNS and similar tasks.

The interested reader can see live stats of the total number of requests to the Squid servers at <http://noc.wikimedia.org/stats.php>. The number of peak requests per second is usually around 5000.

6.2 Deployment scenarios

Wiki technology can be used to edit and present a large number of different types of information. The quick editing enables one or multiple users to quickly do changes and keep the information up to date. The information is published at once, and an archived item is automatically generated.

This versatility and simplicity creates a wide range of possible uses for wiki engines. The information that is written to a wiki can be an unstructured text, or conform to a specific set of rules. The wiki concept is still young and we have only seen the beginning of all possible uses and extensions.

A wiki engine could be deployed in many ways, and the architecture and design should ideally be so flexible that a single implementation can cover all, or most of them. Today wikis are mostly used for public web sites publishing information about a single topic, or as an encyclopedia. The wiki engine is installed for one dedicated wiki site on a general purpose web server.

6.2.1 Wiki on a stick

USB sticks has become very popular, they come in a large variety of different designs and memory sizes. They are often used to transfer files such as documents and images from one computer to another. There is no technical problem with installing programs on a USB stick, in fact it is possible to install for example a Linux distribution and boot a computer from the stick. A memory stick is small enough to always be around, so why not design a wiki engine that can be run from a stick? This type of wiki could be used as a personal information manager, to keep notes, address information, and web links and so on.

What specific requirements does this make on the wiki engine?

- Easy XCOPY installation
- Zero configuration
- As few dependencies as possible
- Able to scale from local use to server mode
- Small disk and memory footprint

Easy XCOPY installation

The installation needs to be as simple as possible. Most wiki engines of today are not easy to install. The implementation could be packed together with a standard XSL-skin, images etc into a single executable file. The wiki engine is installed by just copying this single file onto the stick.

Zero configuration

The default configuration should cover most uses, and to run the engine all the user has to do is to run the exe-file from the stick. The wiki engine is started, an icon in the task tray in Windows XP, or similar area in other operating systems is used to indicate that the wiki server is running. Right clicking on this icon brings up a menu that has options to shutdown the wiki server, to go to the wiki server's homepage and to a configuration page. The application listen for http requests on localhost, port 80, thus the exe implements a minimal web server, similar to the server bundled with Microsoft Visual Web Developer 2005. The default configuration should be a safe mode, where only requests from the local computer is possible.

Dependencies

Everything required to run the wiki engine should be included in the single exe-file. There should not be any external dependencies on installed programs. This rules out the option of using a relational database as backend, instead the simplest possible backend should be used, that is the file system. All active pages and archived pages are stored as XML-files on the stick. This makes it easy to share pages between different users, since they can share a page, or a set of pages by either copying them into another wiki stick, or by emailing them. Backup could be done just by copying the xml-files to a safe location.

Scaling

There should be an option to configure the wiki engine to listen for remote request and thereby act as a web server for request from the internet. The wiki on a stick implementation could then be used as a wiki internet site. If the user then decides that better performance is required, there should be an easy way to move the xml wiki files into a relational database.

Footprint

The wiki on a stick implementation should be designed to require little memory and disk space. It should also be designed for a limited number of request per seconds, thus there is no need for caching of pages etc.

Implementation details

Most of the code of wiki on a stick can be shared with the full blown wiki engine implementation. The parts that are different are mostly the lack of relation database storage and disabling or at least simplification of caching features to decrease memory usage.

Even thus the default installation should have zero configuration, wiki on a stick should have most of the features of the full version, for example customized XSL-templates and images.

6.2.2 Personal wiki server

This is one notch above the Wiki on a stick implementation, the feature set should be similar, but it should include caching and have support for relational database storage. The performance with a relation database as backend will be much better than using the file system.

The wiki engine should either be run with the same included mini web server as the wiki on a stick implementation, or be installed as a windows service running in the background with no user interface except by http. The wiki engine should also be possible to be hosted by a web server.

6.2.3 Wiki hosting

A wiki engine configured to only allow logged in users to edit content can be used as a content management system for personal or company web sites. It is hard to draw a line between the wiki concept, the blog concept and web content management systems, they all share a lot of the same features and as each one is becoming more and more advanced they overlap more and more in feature set.

Blogging have become very popular during the last year, one reason for this is the easy and free creation of a blog at blog providers such as blogger.com. It is a simple registration process and a few minutes later you have your own blog where you can publish information.

Wikis have so far not been used in this way, but in a few years time there will probably be services like this for starting your own wiki. This creates a new set of demands on the wiki engine. Instead of one wiki site hosted on a server with hundred of thousands pages, there will be hundred of thousands wiki sites on a server with a small number of pages each, maybe in the hundred. The caching strategy has to be designed to perform well in both these situations. Configurations and stylesheets have to be cached in a way that gives the best performance possible, without using the cache to store resources on wiki sites that are seldom accessed.

The different wiki sites hosted on a server needs to have their own configuration, their own XSL templates.

The database storage has also to be designed to handle multiple sites, preferably by storing all pages in the same database using shared tables. Advanced relational database servers supports horizontal partitioning where the load can be split between different disks and even different servers.

6.2.4 Summary

A state of the art wiki engine that both supports deployment on an USB stick and on a wiki hosting server requires an architecture and design that scales well. The support of complex hosting environments should ideally be implemented in such a way that the implementation still enables developers to add exten-

sions without having to care about if the extension will be used in a hosting environment or on a stick.

The wiki engine should be a framework, providing simple interfaces that extensions can use, without having to care about different caching scenarios. This type of services should the framework handle.

6.3 Client scenarios

There are many different types of clients used to access web content, everything from Internet Explorer on the Windows platform, to WAP based browsers on a mobile phone. Wiki pages are usually heavy on information in text form, the layout is mostly there to enhance structure instead of layout. This makes wiki content ideal to be transformed into formats and layouts that are optimized for each client type. The use of XML as storage format enables easy transformations with XSLT. The type of clients that is used to access web pages will probably become more and more diverse. The wiki engine should have support to deliver content to all or at least most type of clients. Client can be categorized into three categories:

- Fat clients
- Thin clients
- Special purpose clients

6.3.1 Fat client

Internet Explorer and Mozilla Firefox are examples of fat clients. They have support for (X)HTML, CSS, XML parsing, XSL transformations and JavaScript.

The transformation of the wiki markup into XHTML is usually performed on the server. This transformation takes up server processor resources. Since the fat client can perform this task, it is better to do this step on the client when it is supported. This results in quicker response times, since the server needs to do less for each request, the transfer rate increases if the XML document is more compact than the resulting XHTML document witch should normally be the case. The offloading of this step from the server will enable more requests per seconds.

A single XML document can be transformed into many different HTML representations, depending on which set of XSL transformations that is applied to it. A user should be able to choose between different layout themes. This has the disadvantage that if HTML responses are cached at the server then multiple HTML documents might be cached representing the same wiki page, but with different layouts. If the wiki engine instead serves wiki XML markup, then the cached entries does not depend on the layout, resulting in a better utilized cache and a better cache hit ratio.

Today most sites using XML for content storage perform the transformation on the server, this is simpler since the type of client does not need to be taken into account, and there are no problems with XSLT implementation differences between different browser vendors.

AJAX

Asynchronous JavaScript and XML usually specified with the acronym AJAX is a programming model that is becoming more and more popular in advanced web pages.

A user action that normally would require the whole web page to be reloaded in the browser is replaced with an asynchronous JavaScript call that requests the new data from the server and uses the response to replace the part of the web page that needs to be updated. This enables a richer and more responsive user experience. The request and response is usually in the form of an XML document but can be in any form. (Wikipedia: AJAX, 2005-11-05)

A wiki implementation could use AJAX to:

- Request new wiki pages
- Switch to edit mode
- Provide interactive help during editing

Request new wiki pages

The traditional web navigation model requests a whole new page when the user clicks on a link. There is usually some content that exist on every page on a given site. This is usually the menu, copyright notice and some layout elements that give the pages a common look and feel. For every request the markup specifying this content is resent, this is a waste of bandwidth and results in longer response times.

An alternative to this traditional web navigation model is to use AJAX, one HTML page is requested from the server, and this HTML page contains JavaScript code that handles all subsequent requests from this client session.

When the user clicks on a link, the JavaScript request the corresponding XML document from the wiki server, when the client receives the request, the XML document is parsed and transformed with XSLT, and the result will replace the page content of the old page. The menu and other elements are not changed. The initial HTML page is in fact a browser in itself, a browser of XML wiki content, using the browser as a rendering engine.

Switch to edit mode

The AJAX enabled page should cache the latest XML response, thus if the user choose to edit the page, it is just a matter of transforming the latest XML response with an XSL template designed for editing. No request has to be made to the server, enabling very quick and convenient switching to edit mode.

Provide interactive help during editing

During the editing of a wiki page, AJAX technology could be used to assist the user, it could suggest links to other wiki pages, a type of intelligence for wiki editing. It could also spell check the text, and inform the user if there is an editing conflict.

6.3.2 Slim clients

Not all browsers have support for XML and XSL transformations, some browser runs on hardware with very limited processor and memory resources. An alternative representation is required for this category of clients, the functionality should be the same when possible but the user interface has to adapt.

Normal XHTML navigation or WML should be used, the layout of the page should be adapted to the clients screen sizes. Editing is done in a XHTML textarea, editing the Wiki XML directly. The user experience is not as rich, especially during editing, but probably these devices will mostly be used to read the content, and if edits are done, they will probably be on a small scale.

6.3.3 Wiki web service

A web service is a software system enabling machine to machine interaction over a network. Usually either the SOAP protocol is used where both the request and response are XML messages, or the simpler REST approach where the request is an URL and the response is an XML message. (Wikipedia: Web service, 2005-11-06)

Web services enables different systems to interchange information in a platform and programming language independent way. One example of a web service is Amazon.com that has a rich set of interfaces to access almost all the content on their website through either SOAP or REST. In the REST case, the request URL specifies the query, for example you could ask Amazon for the title and price of all books written by a specific author. (Amazon: Web services, 2005-11-06)

Yahoo and Answers.com incorporates Wikipedia information into their site, this is now probably done by mirroring the Wikipedia database, but it could be performed in real time using web services.

An IT-related company could for example on their website have a dictionary of computer terms, and instead of writing their own definitions, they could request an XML formatted document from a wiki using web services, and format that document into their own layout.

The wiki service could be used not only by other web sites but also by applications. One example of this is a text editing program that could use a wiki web service on a public wiki dictionary to request synonyms.

The AJAX DHTML browser and editor discussed above would be a wiki web service consumer, requesting XML wiki documents and perform their own transformations.

Because machine to machine requests can easily create large amounts of traffic, care have to be taken both at the wiki web service end, to be able to control the number of request from a particular consumer, and the consumer should probably implement some caching strategy for frequently accessed data. Amazon web services requires each consumer to be registered and to use an identifying ID for each request. Amazon limits the number of request for a specific ID to one per second.

6.3.4 Search robots

The free text search functions on wikis are often quite limited, instead Google or some other search engine is used to get the information. The high number of links between pages, and from other sites to a wiki site, is favorable in the search engine ranking of pages, and often Wikipedia and other wiki sites are returned near the top of the search result.

It is probably not a good idea to index all pages contained in a wiki, the discussion and archived pages should be excluded from the indexing.

The wiki engine should detect search robots and serve pages optimized for good page ranking and easy indexing.

6.4 Programming language

The technologies and ideas in this chapter is mostly programming language neutral, but to be able to architect and design for optimal performance the strength and weaknesses of the implementation language have to be taken into consideration.

PHP

Most wiki engines today, including MediaWiki, are implemented in PHP. PHP is an open source server side scripting language. This acronym has had different meanings, from the beginning it stood for *Personal Home Page* but today's official meaning is the recursive acronym *PHP Hypertext Preprocessor*.

PHP is widely adopted and is one of the most popular server side scripting languages, in May 2003, almost 13 million domains were using PHP. The large set of free and open source libraries that is included in PHP is a major advantage for PHP compared to many other scripting languages. It has built in support for XML and XSLT.

The latest version, PHP 5, has enhanced support for object oriented programming, and has support for reference type objects, private and protected member variables, abstract, private and protected methods, and abstract classes. It also has support for interfaces and cloning.

There is no native support for Unicode, but this is planned for either version 5.5 or 6.0.

PHP is a dynamically typed language, variables do not have to be declared. This is often something appreciated by beginner programmers, but as the complexity increases it often something that makes debugging harder. (Wikipedia: PHP, 2005-11-06)

6.4.1 Java

Java is a reflective, object oriented programming language developed by Sun Microsystems. It is popular to use for large enterprise systems, but can scale down to desktop applications and mobile phones. Java programs are platform independent and require the Java Virtual Machine to run.

The five primary goals in the creation of the java language were:

- It should use the object-oriented programming methodology.

- It should allow the same program to be executed on multiple computer platforms.
- It should contain built-in support for using computer networks.
- It should be designed to execute code from remote sources securely.
- It should be easy to use and borrow the good parts of older Object Oriented languages like C++.

The latest version of Java, 5.0 has support for generics, class and method metadata, enumerations and includes an integrated XML parser and XSLT processor. (Wikipedia: Java programming language, 2005-11-06)

6.4.2 C#

Microsoft developed C# as part of the .NET platform. C# has strong similarities with C++ and Java. According to some people it is a direct copy of Java, but there are many differences. Microsoft studied the first years of Java usage and create a language that took the best parts from Java, and redesign the parts that programmers did not like with Java. C# compiled code is running on the .NET platform, which theoretically is platform independent.

C#, VB.NET and other .NET languages all share the same framework library, the .NET Framework. The .NET framework includes support for a large number of usage scenarios, and has extensive support for XML and XSLT processing.

The 2.0 version of the .NET framework and C# has just been released at the time of writing. The 2.0 version has support for generics, and includes a lot of performance enhancement, especially in the XML area.

There are several open source projects that are creating their own implementations of the ECMA-334 C# specification and the .NET Framework. (Wikipedia: C Sharp, 2005-11-06)

Mono

Mono is an open source project, led by Novell, to create a set of .NET compatible tools, including a C# compiler, the Common Language Runtime and the .NET Framework. Mono enables .NET application to be run on Linux, Solaris, Mac OS X, Windows and UNIX. The project was announced in July 19, 2001 and three years later, Mono 1.0 was released. The current stable version is 1.1.9.2. (Mono, 2005-11-06)

The major parts of the .NET framework are complete but there are still many areas not yet complete for full compliance with MS .NET Framework.

DotGNU

DotGNU is another open source implementation of the .NET platform including the C# language. It is sponsored by GNU project, which is supported by the Free Software Foundation. (DotGNU, 2005-11-06)

6.4.3 Performance

It is hard to compare different computer languages when it comes to performance, often the tests are written by, or sponsored by the company behind the language that wins the test. It is vital that the tests when implemented in each language are as optimized as possible.

Java Pet Store is a blueprint application developed by Sun to showcase best practices. The same application was later implemented in C#. The C# implementation required just 3 484 lines of code, while the Java implementation had 14 273 lines of code. This is an indication that the .NET Framework library supports web applications better than the Java library. The .NET implementation was around twice as fast as the Java implementation, and the .NET implementation scaled better when the number of concurrent users increased. (VeriTest, 2005-11-06)

The wiki implementation will greatly use XML, thus the performance of the XML parsing is of vital importance. Sun conducted XML benchmarks between JVM 1.4 and .NET 1.1 using their own benchmark application XML Mark. The result of this benchmark was that Sun concluded that the Java platform significantly outperformed the .NET platform.

Microsoft has released new benchmarks using .NET 2.0, and they have also fixed two issues with the original XML Mark version, that missed to serialize the XML Test document to disk when using Java, and that did not use an optimal way of getting elements in the .NET implementation.

The benchmark shows that the XML support in version 2.0 of the .NET Framework has been greatly enhanced, the performance when using the XML DOM to parse an XML document is threefold better in 2.0 compared to 1.1, and about the double compared to Java JVM 1.5. When using a SAX parsing model, .NET 2.0 is also significantly faster than Java. (MSDN: Comparing XML Performance, 2005-11-06)

6.4.4 Choice of programming language

There are multiple factors to take into account when choosing an implementation language. PHP is a script based language, and loosely typed, and besides since the MediaWiki software already exists implemented in PHP, the choice is between Java and C#.

The following factors were considered:

- Performance
- Rich XML support
- Libraries
- Future development

Performance wise C# seems to have the edge over Java, both the XML benchmark and the web application benchmark shows that C# is able to handle a larger user load and still serve more requests per seconds than Java.

Both Java and C# have rich support for XML. The Apache Foundation has a free XSL-FO processor for Java, this enables XML documents to be transformed to PDF and SVG (Apache: FOP, 2005-11-06). This could be very useful for

some of the features previously discussed. There are XSL-FO processors for .NET, but at the moment all of these seem to be commercial.

Both Java and C# have large libraries, Java has the advantage of more free software support, .NET has the advantage of usually simpler interfaces as can be seen by the required line of codes for the Java Pet Store implementation.

Java is multiplatform, java server applications can be run on a large number of different hardware platforms and operating systems. The .NET platform is also multiplatform, but so far the support for other platforms is heavily under development.

C# was chosen as the development language for the experimental wiki implementation described in this thesis, and the rest of this chapter will sometimes contain C# specific considerations.

There are not many current wiki implementations written in C#, thereby it is interesting to create one, and if this wiki implementation sometimes will be released under an open source license, it will probably get more attention than another PHP or Java wiki implementation.

6.5 Storage system

The storage system must be flexible and scale from small personal wikis with megabytes of data, to large wikis with several hundred gigabytes of data.

Traditionally small systems have used the file system for storage, this works well for small amounts of information. It is simple, it works with *Wiki on a stick* and content, in the form of individual XML files, can easily be shared and backups are done by just copying the files. Depending on what file system the disk is using, different technical limitations have to be taken into consideration.

A large wiki site, with several hundred of gigabytes of content requires a more robust and flexible storage system. The storage system must be able to fulfill the following requirements:

- Storage of XML documents
- Quick access to document based on key
- Support for scale up, and scale out
- Data integrity
- Backup

The only storage system that supports all these requirements is a state of the art relational database with XML support. There are many different database vendors, and the implementation should be designed in such a way that it is easy to change the database layer of the application, and thereby enable a change of the database engine without any required change to other parts of the implementation.

6.5.1 Performance of database servers

There is a non-profit corporation called TPC that define transaction processing and database benchmarks. There are different set of benchmark depending

on which type of application that is simulated. There is also different ways to present the result, the most interesting is to look at the price/performance ratio. There are two different popular benchmarks, TPC-C and TPC-H, both of them have Microsoft SQL Server at the top of the price to performance charts. (TPC.org, 2005-11-06)

6.5.2 MS SQL Server 2005

There is a new version of MS SQL Server that is just released, it is the 2005 version, succeeding the widely used 2000 version. The new version includes key enhancements to data management in the following areas:

- Manageability
- Availability
- Scalability
- Security

(Microsoft SQL Server 2005 Product Overview, 2005-11-06)

Manageability

The support for managing large databases with hundred or thousands of tables and stored procedures are greatly increased with the new SQL Server Management Studio. All administrative tasks can be done through this integrated management console.

Availability

MS SQL Server 2005 has better support for database mirroring, failover clustering and database snapshots. These are technologies that should be used in a large wiki server setup to minimize downtime.

Scalability

Large tables can be horizontally partitioned into file groups. This enables for example old pages that are not so frequently accessed to be stored in another file group that can reside on slower and cheaper disks. The database will still be functional even if not all the file groups are available, so in a restore scenario, the active data could first be restored, and when the wiki site is up and running, the restore could continue with archived pages that are not necessary for normal page viewing.

MS SQL Server 2005 has support for 64-bit Itanium2 and x64 systems. The server can thus address larger amount of RAM.

Security

The largest enhancement to security is that the default installation is a secure configuration with all optional features disabled. It has also support for native encryption, using a key infrastructure.

Different editions

Microsoft SQL Server 2005 is released in several different editions, the SQL Server 2005 Express is a free desktop edition, it support almost all of the features, but is locked down in terms of processor and memory, supporting 1 GB of RAM and a maximum database size of 4 GB and only one processor.

The Express Edition is great to use for small installations and for development of SQL Server applications without a license to any of the none free editions.

Since the editions work in the same way, the application does not need to be changed when scaling up from the Express Edition to the Enterprise Edition.

XML Support

The most important new feature for a wiki implementation in Microsoft SQL Server 2005 is the native support for an XML data type. In earlier versions an XML document had to be stored as a binary large object or as a text data type, but with version 2005 a column can be declared to have the XML data type.

XQuery, or rather a subset of the XQuery standard is used to request specific data from an XML column. Special indexes can be defined on XML columns to enhance query performance.

This enables some of the logic to be moved from the business tier into the database tier. This is a new feature and there will probably take some time before best practices are established on how to utilize XML data inside a relational database. Until otherwise proven, care should be taken to not over utilize these features.

Choice of database servers

There are other database servers having similar support to XML as Microsoft SQL Server 2005, but due to previous familiarity and developing experience by the author on Microsoft SQL Server 2000, and due to the fact that it is known as a high performing relational database server and that it has a free express edition that can be used on personal wiki sites, the choice for a database server to use for the wiki implementation in this thesis is Microsoft SQL Server 2005.

6.6 Performance versus simplicity

There is always a tradeoff between performance and simplicity. Simple system can sometimes be efficient but in the case of server applications good performance is often depending on good caching strategies making the implementation more complex.

The support for wiki hosting, that is, multiple wiki sites should be able to run on the same server process and to have the same database as storage is also a design decision that greatly increases the complexity.

The design and development have to be influenced by the performance and scalability goals at every point, but the optimization should be where it gives the largest payback, to optimize functions that are seldom used is not important, instead the focus should be on the most frequent operations.

The processing pipeline should be flexible but still very effective.

The .NET framework documentation is mostly about how the different features should be used, not very much is mentioned about the performance difference between different design choices. Some parts are obvious, like using the right data structures, but when it comes to different types of XML processing, what parsing methods to use etc, then there are two ways to find out the optimal solution. The first one is to code all alternatives, and then try them out, collecting benchmarks, and then choose the method that was the quickest. This is a very slow method, and can be error prone if the benchmarks are conducted under conditions that differ from the real server usage. Another way is to study the source code, often a method can accept many types of different inputs, say a stream or a XML DOM document, by reading the source code it is sometimes possible to see ways to increase the performance.

The source code of the commercial version of .NET is not available, there is one other version released by Microsoft that has publicity available source code. But since this version is not guaranteed to be the same as the commercial version, it is much better to use a reflector tool.

6.6.1 Reflector for .NET

Lutz Roeder has written a reflector tool for .NET. It has an integrated class browser and IL, Visual Basic, Delphi and C# decompilers. The reflector tool can be used on all managed code in the .NET Framework, enabling anyone to exactly see how a class is implemented. (Lutz Roeder, 2005-11-06)

This is helpful when dealing with poorly documented code, or when the programmer wants to know specific details about the implementation. The callee graphs are useful to find class dependencies and to get a good overview of what other classes uses a specific class.

6.7 Scale out versus Scale up

There are two different ways to scale an application:

- Scale up
- Scale out

6.7.1 Scale up

Scale up is basically to beef up the server, by adding more memory, more processors, larger disks and more network adapters. The application should be run and metrics should be gathered to find out what resource the application is bound by. If the CPU processing power is the bottleneck, then more CPU's should be added and so on.

This is usually the easiest way to scale an application, it is simple, as long as the application is multithreaded it will use the extra processors, and it can be a cost effective way to scale an application.

There is still the problem of one point of failure, if the server goes down, the application goes down. (Meier et al., 2004, chap. 3)

6.7.2 Scale out

Scale out is to add more servers, instead of running the application on a single box, it is distributed on several boxes. Depending on the architecture of the application, it might be enough with one shared database server between multiple web servers. If the database is the bottleneck, the database could be partitioned over several database servers, or distributed to several servers with replication. (Meier et al., 2004, chap. 3)

6.7.3 Loosely coupled and layered design

By using a clean, loosely coupled, layered design it's much easier to scale out than with tightly coupled layers. Web applications usually have two very distinct layers, the web server part, and the database server, this is one natural layer boundary.

Stateless components scales out better than statefull, since they do not have any in process-state across requests. This makes it easy to deploy the application in a web server and to use a simple load balancing solutions between the different web servers.

Microsoft and others give the following recommendations for good application scalability:

- Design for loose coupling
- Design for high cohesion
- Partition application functionality into logical layers
- Use early binding where possible
- Evaluate resource affinity

(Meier et al., 2004, chap. 3)

6.8 Design patterns

Software engineering contains tasks and problems that are similar between different projects, instead of reinventing new solutions each time, best practices arises and these best practices has been formalized as design patterns. A design patterns is a general solution to a common problem. By using well know design patterns in an implementation, other developers can quickly understand how the application is structured. Design patterns are often published in books and on web sites. A design pattern is no hard set of rule that has to be implemented in a certain way, instead it is more of a guide, and the pattern can be adapted to fit the particular need.

6.8.1 Adapter pattern

The adapter pattern is a very common pattern, the book Larman (2002) defines it as:

Adapter

Context/Problem: How to resolve incompatible interfaces, or provide a stable interface to similar components with different interfaces?

Solution: Convert the original interface of a component into another interface, through an intermediate adapter object.

One frequent use of the adapter pattern is to create a stable interface between the business layer and the database layer, enabling the implementation to switch between different database layers without any code changes in the business layer.

6.8.2 Provider pattern

Microsoft has as part of the .NET Framework 2.0 defined a pattern called the provider pattern. The provider pattern is designed to solve three problems:

- Fixed choices
- Fixed internal behaviors
- Fixed data schema

(Howard, 2005-11-07)

Fixed choices

APIs usually contain a fixed number of choices, one API could have support to store data in a SQL Server database or in the file system, but if the user wants to store the data in an Access database? Often the only solution is to not use that API and instead reimplement the functionality in a custom class.

Fixed internal behaviors

The internal behavior of an API is usually fixed, sometimes it is possible to inherit from the class and override specific members, but often the developer can not control the API to use a custom class instead of the default class, especially if it is a class used inside the API.

Fixed data schema

Often the database or xml schema is fixed in an API, if the developer wants to save the information into an existing database table, there is no way to specify a custom mapping.

The usual fix

The usual way to come around the above problems, is to not use a published API, instead the developer implements their own API, having the features required. This has the drawback of a lot of extra work, instead of just changing the few things that need to be changed, the whole API is reimplemented. There is also an advantage to use published API since they are well known to other developers, making it easier for other to change your code.

A better solution

Microsoft faced the above problems with many of their APIs, the solution they came up with was called the provider pattern.

The pattern itself is exceedingly simple and is given the name "provider" since it provides the functionality for an API. Defined, a provider is simply a contract between an API and the Business Logic/Data Abstraction Layer. The provider is the implementation of the API separate from the API itself. (Howard, 2005-11-07)

What does this mean? Basically, the developer can specify in a configuration file, what classes the API should use internally. The Microsoft provider pattern specification, has a long list of requirements, all providers should inherit from one specific ProviderBase class, and have a set of required properties. The interested reader can read the Microsoft definition of their provider pattern for more information.

6.9 Caching

The use of good caching strategies can greatly improve the performance of a server application. It is important to cache the right data. Data that is inexpensive to create should not be cached, instead the focus should be on application wide data and data that is used by multiple users that are expensive to create or retrieve. For data that is requested very often, even caching data for just a few seconds can greatly enhance performance. A web site with thousands of hits per second, will probably have a few pages that are much more frequently accessed than others, even if these few pages, like the first home page, have data that often change, there is probably a good idea to cache the response and use that for the next seconds of requests, even if that means some user will get an out of date version of the page, a page that is a few seconds old, will probably not be a big problem.

The cached data should be in the appropriate form, avoid if possible to cache data in a form that needs to be further reprocessed.

6.9.1 Database cache dependency

Microsoft .NET Framework 2.0 and Microsoft SQL Server 2005 has support for database cache dependency. This means that the application will be notified when the cached data is changed in the database. This enables the application to remove data from the cache when it has become changed in the database, or to repopulate the cache with the new data.

6.9.2 IIS and ASP.NET

Both Microsoft Internet Information Server and ASP.NET has support for several advanced caching strategies. This can be used to cache the whole response of a specific URL, or to cache the parts of a page that is static between requests. Using these features will create a dependency on the Microsoft ASP.NET implementation and on IIS.

6.10 CSS stylesheets

The output of the wiki engine will result in an XHTML document, the transformation take place on the server for clients that do not support XML and XSL transformation. The XHTML standard is much more about defining structure than HTML was, elements that before defined layout, such as the font tag have been removed and instead CSS stylesheets should be used to specify the layout.

CSS stylesheets can greatly change the visual appearance of an XHTML page without any changes to the XHTML code. One good example of this is the website <http://www.csszengarden.com/> that contains a large set of layouts based on the same XHTML code, but with different CSS stylesheets.

To make CSS as useful as possible it is important to write XHTML code that is CSS friendly, to specify class, and ID attributes on elements.

In a wiki hosting scenario, where thousands of wiki sites are hosted on the same server, CSS might offer a rich enough functionality to customize the different sites visual appearance.

6.11 XSL stylesheets

XSL can be used to transform the XML wiki documents into XHTML. XSL or more specific XSL Transformations, were designed for the purpose of transforming an XML document into another XML document. The wiki engine should be able to serve content to different types of clients. For each specific client type, a set of XSL stylesheets can be used.

It should also be possible to have different skins, or layout themes, that the user or administrator can choose from. These skins should be shareable between different wikis hosted on the same server, and ideally only one copy should be kept in the stylesheet cache.

The .NET Framework 2.0 adds support for compiled XSLT stylesheets with the class `XslCompiledTransform`. The execution times are on average 4 times better than the old `XslTransform` implementation.

The `Load()` method reads the stylesheet and creates an abstract syntax tree. The abstract syntax tree is then compiled to an assembly. The `Transform()` method builds an XML tree for the input document, if the input document was specified as a class implementing `IXPathNavigable`, the `CreateNavigator` method is called and use the returned navigator as it's document input cache.

There are various methods to retrieve the result of the information, they are all explained in detail in the MSDN blog by Dubinets and Lapunov (2005-11-07).

6.12 Preprocess

The access pattern to a wiki engine contains a majority of read request compared to edit request. The most common request is to read the latest version of a specific document. The document can contain XML elements that are replaced by dynamic content. There should be support to create extensions that for example parses the content of an `<isbn>0-87773-860-2</isbn>` element and get the related book information from a web service like Amazon. This kind of active content does not need to be evaluated on a per request basis, instead this information can be cached. Different information can be cached for different

amounts of time until it becomes out of date. The book title is an example of data that never will be changed and can thus be cached indefinitely. An element like `<stock-quote>MSFT</stock-quote>`, that should be replaced by the current trade quote of the Microsoft stock, can not be cached for a long time until the data is out of date. As discussed earlier, even if the duration of the caching is short it can enhance the performance greatly for frequently requested pages.

All transformations and data operations that are not unique to a certain request should if possible be cached and the result reused for the following requests. This caching can take place at the web server level, or at the database level.

6.12.1 Web server level caching

If the information is cached on the web server level, the same information will be cached on several web servers in a web farm environment. This offloads the database server since it will not be accessed, but if the dynamic data is expensive to get, or if the web servers do not have the resources to cache the data for a long period of time, due to memory constraints, it will put a big workload on the web servers. This type of expensive data to get, or process, and that can be cached for a longer time than the web servers can cache the data, are good candidates for database level caching.

6.12.2 Database level caching

The previous ISBN example is a good example for data that is better cached on the database level, since the title does not change, the title should be cached at the database level. The title will only be requested once, and then stored in the database. When the web servers process this document, the ISBN is already replaced by the title and no further action on this element needs to be done. Database level caching is no substitute for web server level caching, the web servers should still cache the transformed result (in XHTML) of the request, and use that for following requests.

6.13 Database optimization

Microsoft SQL Server 2005 has support for a large array of tools and technologies to optimize data access.

This section will only go through some basic optimization techniques, database optimization is a too large subject to cover in detail here, some further information can be found in the implementation chapter, but the interested reader should get in depth information from a book specialized on this topic.

6.13.1 Table layout

The layout of the tables should as much as possible follow at least the first three normal forms, but sometimes it is necessary to break against these rules to get better performance. The table layout used to store the XML wiki document will be simple, and thereby it is not that important to create a non-redundant data structure, instead the focus should be on performance.

A large part of the data in the database will consist of archived pages, that are only accessed when a user request an old version of a document. To increase cache hit ratio in the database, the active pages should be saved in a different table than the archived pages. Thus the data pages saved in SQL server will have a higher probability to contain frequently accessed documents.

6.13.2 Indexes

Indexes are used to speed up data lookup, instead of having to scan through all the rows in a table, different types of indexes are used, greatly enhancing the performance. Microsoft SQL Server 2005 contains a tool to analyze a set of database queries and suggest the optimal set of indexes. The primary key or keys in the tables should be chosen to be as small as possible to enable dense indexes.

6.13.3 Stored procedures

To archive the best query performance and a good layered design, stored procedures should be used for all queries against the database. This enables the server to cache query plans, and also enables the database administrator to change and tune queries without having to change the application.

6.13.4 XML Storage

To test the performance of storing XML data in different data types, a set of benchmark tests were implemented in C#, the test were run against Microsoft SQL Server 2005 Express Preview, running on a Virtual PC with 512 MB of RAM.

The benchmark program run each test first once, to populate the cache at the SQL Server, then three runs that were timed, and an average execution time was calculated. The test retrieved random rows from a table with 10 000 rows, the XML documents retrieved were about 14 KB each.

The queries were in the form of stored procedures, two different return types were tested, either as an output parameter, or as a scalar value (a recordset returning just one value).

The datatypes that were tested were `NVARCHAR(MAX)`, `VARBINARY(MAX)` and XML. The first run of the benchmarking produced these results:

XML 36,5 seconds

`NVARCHAR(MAX)` 42,5 seconds

`VARBINARY(MAX)` 41,6 seconds

The result between returning the result as an output parameter or as a scalar value did not differ and thus only the output parameter result is presented. The interesting fact is that the XML column was the quickest. The XML datatype is not internally saved as a long text string, instead it is saved in an optimized way for queries. It is strange that the request time is quicker than for a string. This was tested on the preview version, using .NET Framework beta 2, so the result could differ on the final released version.

When examining the benchmark, it was discovered that the XML datatype stripped the whitespace in the XML document, thus returning 13 945 bytes, instead of 14 328 bytes, could this difference in size explain the better performance? The benchmark program was changed to strip the whitespace before the insertion into the database, creating return values of equal length for all column data types. The results were still the same.

During the benchmarking, the file IO was the bottleneck. The benchmark application was changed to select the same row over and over again, ensuring that the relevant data page was in the database server's memory cache. The new benchmark returned the following results:

XML 17,9 seconds

NVARCHAR(MAX) 20,9 seconds

VARBINARY(MAX) 21,2 seconds

The result this time was almost twice as fast as the two previous runs, a clear indication that the disk was the bottleneck in the two previous benchmarks. For some reason the XML column is still the quickest, it is hard to find an explanation to this, but the important thing to notice is that it is at least as quick as storing it as a binary object or as a text string. Since the XML datatype enables XML indexes and XQuery and seems to have better performance, there is no reason to not use the XML datatype.

6.13.5 Replication

Replication enables data from one server to be copied to another server. Different kinds of replication can be configured, the simplest is to have one master database that is used to do all the changes to the data, they are then copied over to slave databases that are only used for retrieval of data. A more advanced, and more failsafe setup involves merge replication, where data is send in both directions between servers. This introduces problems of merge conflicts, and precautions have to be made to solve this.

Preferable the database design should from the beginning support different types of replication.

Chapter 7

Implementation

The wiki engine implementation has the code name KuaiWiki. Kuai is Mandarin and means fast, quick, quick-witted, ingenious, sharp or straightforward. This chapter describes some details of the implementation.

7.1 Prototype in .NET Framework 1.1

The first step was to implement a prototype to learn more about the problems related to a wiki engine. The prototype was developed in .NET Framework 1.1 and C#. The prototype was designed to be very flexible and used a list of delegates. A delegate is similar to a function pointer in C++. The delegates were executed in turn, and the result from one delegate was the input to the next delegate in the list. The idea was to create a pipeline that easily could be extended, and where steps easily could be changed. The list of delegates was created based on a configuration file that contained assembly names, class names and the method to call.

It soon became very obvious that even if this design allowed great flexibility that it needed more structure, not every feature was possible to fit into a flat pipeline.

The database returned multiple XML documents for each request, at least two, one containing the article and one containing *what links here* data. The *what links here* data is a list of the articles that have links pointing to the current article. The *what links here* data was selected by a database query that selected all links each time an article was requested. These two XML documents had to be merged, and interception elements, which will be described later, were replaced with their related data. The retrieval of several XML documents for one article was found to be an easy way to get the information that was needed, but two database queries are slower than one. It was decided to change this part of the prototype to the implementation version, to enable more logic to reside in the database, and to preprocess as much as possible, thus it would only be necessary to return one XML document from the server for each article request.

The prototype had served its purpose, the lessons learned while developing it led to a new design used in the implementation.

7.2 .NET Framework 2.0

During the development of the prototype, the .NET Framework 2.0 beta 2 and Visual C# 2005 Express Beta 2 were released. After some initial testing, it was decided that the implementation should be written in .NET Framework 2.0 using the new features found in C# 2.0.

The new features that were most interesting for a wiki implementation were:

Generics - also called parameterized types or parametric polymorphism. This has some similarities to C++ templates. Anders Hejlsberg the lead C# architect explains the implementation details in an article at <http://www.artima.com/intv/generics.html>

New collection classes - the new collection classes uses the generics feature. It is possible to define collections containing objects of a specific type, thereby the casting from type Object to the specific type is no longer required.

Improved performance - Microsoft has focused on improving the performance throughout the framework.

Better XML support - a new XslCompiledTransform class that greatly increases the performance for XSL transformations.

7.2.1 SQL Server 2005

A preview of Microsoft SQL Server 2005 was also released during this time, and after some testing it was decided to use the new XML features and design the wiki engine primarily for Microsoft SQL Server 2005.

7.3 Modified provider pattern

The KuaiWiki implementation contains around 60 classes, most of these classes implement a modified provider pattern. Instead of specifying in the code the classes that should be used, the implementation looks in a configuration file.

One example of this is the page source functionality found in KuaiWiki. There is an abstract class called `PageSource` that specifies two methods, `TryOutput` and `TryInput`. The `KuaiWikiMainPageSource` inherits from the abstract `PageSource` and provides implementations for the `TryOutput` and `TryInput` methods.

There is also a `HelloPageSource`, this is a simple class that returns different *Hello World!* messages. This class is designed to test different features in the framework. The `HelloPageSource` also inherits from the abstract `PageSource` and implements its methods.

The configuration file contains the following XML fragment:

```
<PageSources>
  <PageSource active="true" assembly="KuaiWiki"
    class="KuaiWiki.PageSources.HelloPageSource">
    <Message>Hello Config!</Message>
  </PageSource>
```

```

<PageSource active="true" assembly="KuaiWiki"
  class="KuaiWiki.PageSources.KuaiWikiMainPageSource">
  <Connection>Server=192.168.1.1,1433;UserID=Kuai;
    Password=f3d20v;Database=KuaiWiki</Connection>
  <SiteId>1</SiteId>
  <Interceptors>
    <Interceptor active="true" assembly="KuaiWiki"
      class="KuaiWiki.Interceptors.Interceptor"/>
  </Interceptors>
</PageSource>
</PageSources>

```

The configuration fragment above specifies two page sources, the `HelloPageSource` and the `KuaiWikiMainPageSource`. Both the assembly name and the full type name are specified, enabling the classes to reside in another namespace and another assembly.

Both page source elements have an `active` attribute that can either be true or false, if it is false the element is ignored. When the implementation needs the page source objects, the above XML fragment is sent as an argument to the `WebConfig` class that contains the logic to parse and instantiate the correct objects.

The reflection feature in C# makes this very easy, and requires just a few lines of code:

```

public object GetInstance(XmlElement xmlElement) {
    string assemblyName = xmlElement.GetAttribute("assembly");
    string className = xmlElement.GetAttribute("class");

    Type type = Type.GetType(className + ", " + assemblyName);
    MethodInfo method = type.GetMethod("GetInstance",
        BindingFlags.Static | BindingFlags.Public);
    if (method == null) {
        throw new NotSupportedException(
            "Missing GetInstance: " + type.FullName);
    }
    object[] parameters = new object[] { this, xmlElement };
    return method.Invoke(null, BindingFlags.Default, null,
        parameters, null);
}

```

The classes are required to have a static method `GetInstance` that accepts parameters of the type `WebConfig` and `XmlElement`. The `WebConfig` is the class that instantiated the object, this is used to get access to various system wide properties. The `XmlElement` object contains the configuration that this specific object instance should use. In the above `HelloPageSource` configuration example, this is the `<Message>Hello Config!</Message>` data.

The `KuaiWikiMainPageSource` has a more advanced configuration section, where the connection string to the database is specified together with the `SiteId`, but it also contains the `Interceptors` element which contains information about another class implementing the modified provider pattern. This pattern, not

only enables to initialize classes based on information in the configuration file, but it also allows nested providers. This is a major improvement compared to the first prototype implementation.

The modified provider pattern allows configuration information specified as XML elements, and also provides nested providers. If a developer creates a new version of the `KuaiWikiMainPageSource` class, he can compile that into a new assembly, and publish it on the internet. A KuaiWiki administrator, can download the assembly and change the line:

```
<PageSource active="true" assembly="KuaiWiki"
  class="KuaiWiki.PageSources.KuaiWikiMainPageSource">
```

Into:

```
<PageSource active="true" assembly="ImprovedKuaiWiki"
  class="KuaiWiki.PageSources.KuaiWikiMainPageSource">
```

And the KuaiWiki engine will use the `KuaiWikiMainPageSource` class from the `ImprovedKuaiWiki` assembly. The new class will still be using the same `Interceptor` class as before.

7.4 Processing steps

An HTTP request is received by the web server, this request is after some initial processing passed on to the KuaiWiki engine. The result of the processing is usually a returned XHTML or XML document but can also be a binary stream.

Many steps need to be performed, and they vary depending on the type of request.

The wiki engine accepts requests to the static method `KuaiWiki.Process(Request request, Stream result)`. Internally this method uses the singleton pattern to send all requests to the same instance of the `KuaiWiki` class, for an overview of the processing, see figure 7.1.

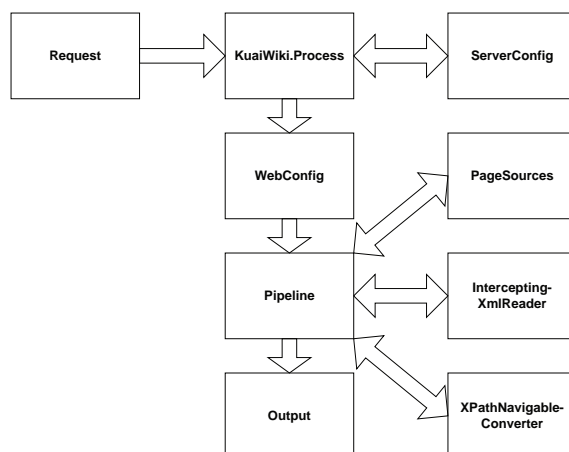


Figure 7.1: Overview of KuaiWiki processing

Most classes are designed to be stateless and to be multithreading safe. This results in better performance, since the objects does not need to be created and garbage collected for each and every request.

7.4.1 Request class

The `Request` class contains all the information that KuaiWiki needs to process a request. It contains URL information and HTTP post data. The class is used instead of the `System.Web.HttpContext` class that is part of the .NET Framework. This is to be independent on any classes in the `System.Web` namespace. This enables the implementation to be run from any web server that can call a .NET class. It also enables easy debugging, since a `Request` object can easily be created in the code, whereby a `HttpContext` object is hard to create due to the arguments to its constructor.

The `Request` class contains properties that are specific to KuaiWiki, and enables quick and efficient parsing of the request URL.

Possible improvements

The `Request` class does not yet contain logic to handle file uploading and cookies. There are also some web server properties that could be useful to be able to access. This could be added in a later release.

7.4.2 WebConfig

The `KuaiWiki` class uses a `ServerConfig` object to get a `WebConfigFactory` object. The `WebConfigFactory` has one purpose, to cache and create `WebConfig` objects.

The `WebConfig` object contains the entire configuration specific to a certain wiki web. In a hosting scenario, one `KuaiWiki` engine can host multiple wiki webs, and each of them have their own `WebConfig` object that is created based on the information in the related wiki web configuration file.

Possible improvements

The current version always use `FileWebConfigFactory`, this should be configurable with the same provider pattern as used elsewhere, but the configuration information should come from a configuration file related to the current `KuaiWiki` instance. There is an abstract class `WebConfigFactory` that is used by the rest of the code, so it is only `ServerConfig` that needs to be changed. A wiki hosting scenario could have thousands of wikis, and it would then be better to have a `SqlWebConfigFactory` that gets the configuration files from a database.

`FileWebConfigFactory` does not implement any caching of `WebConfig` objects. The `WebConfig` class is designed to be stateless and multithreading safe and can thereby easily be cached.

7.4.3 Pipeline

The `WebConfig` object creates a `Pipeline` object, the type of the object is specified in the configuration file.

```

<KuaiWiki>
  <Pipelines>
    <Pipeline active="true" assembly="KuaiWiki"
      class="KuaiWiki.Pipelines.StandardPipeline">
      ...

```

This enables the possibility to change one of the most fundamental parts of the KuaiWiki engine. To test the performance of two different `Pipeline` implementations, two wiki web configurations could be created, where the only difference is the `Pipeline` object. A web server benchmarking tool could then be used to test the performance. This requires no changes to the KuaiWiki engine.

The `StandardPipeline` class performs three major steps:

1. The data provided in the `Request` object is sent to the `TryInput` method on all `PageSource` objects. The request could for example be to add a new article to the database.
2. The `TryOutput` method is called on the `PageSource` objects, if the method returns true, the pipeline continues to the next step, if it return false, the next `PageSource` object is called.
3. The output is transformed to XHTML.

```

public override void Process(
    Request request,
    WebConfig config,
    Stream result)
{
    for (int i = 0; i < _pageSourceList.Length; i++) {
        _pageSourceList[i].TryInput(request, config);
    }

    Output output = new Output(result);

    for (int i = 0; i < _pageSourceList.Length; i++) {
        if (_pageSourceList[i].TryOutput(request, output, config)) {
            break;
        }
    }

    if (!output.IsValid) {
        _errorHandler.Report(500, "Error...", output.OutputStream);
    }

    if (output.XmlStream != null) {
        _xmlConverter.Convert(output.XmlStream, request, output);
    }

    if (output.XmlString != null) {
        _xmlConverter.Convert(output.XmlString, request, output);
    }
}

```



```

}

if (output.XPathNavigable != null) {
    _xpathNavigableConverter.Convert(request,
        output.XPathNavigable, output.OutputStream);
}

if (output.OutputStream.Position == 0) {
    _errorHandler.Report(500, "Erro...", output.OutputStream);
}
}

```

The `Output` class accepts four types of data:

XML String - a string containing an XML document.

XML Stream - a `Stream` containing an XML document.

IXPathNavigable - an object implementing `IXPathNavigable`.

OutputStream - a `Stream` of bytes directly returned to the client without any further processing. This is useful for returning binary files such as images to the client.

The classes inheriting `PageSource` has the option of returning data in any of these four types.

Evaluation

The `Pipeline` class and the implementation in `StandardPipeline` were changed several times until this simple but efficient design was chosen. It is possible to design `PageSource` classes that return none wiki content, and this is something that can be useful as an extension mechanism for more advanced dynamic pages not supported in the `KuaiWikiMainPageSource`.

One example is user registration that could have its own class inheriting from `PageSource` that contains the required logic to register and login a user.

Possible improvements

- The `StandardPipeline` should have better error handling.
- The `PageSource` abstract class could be changed to just include the `TryOutput` method. The `TryInput` method could be specified with an interface, thus only classes mark with the `TryInput` interface needs to be processed in the `TryInput` loop.
- There is currently no caching in this step, page level caching should be incorporated here, or inside the `PageSource` classes.

7.4.4 KuaiWikiMainPageSource

The `KuaiWikiMainPageSource` class handles request for wiki articles. The majority of all request will use this `PageSource`. The `Request` class contains properties to extract prefix, action and some other properties related to a request. The `KuaiWikiMainPageSource` requires that the prefix is `kuaiwiki`.

```
public override bool TryOutput(
    Request request,
    Output output,
    WebConfig config)
{
    if (request.Prefix != "kuaiwiki") {
        return false;
    }
    ...
}
```

The most common action is to request the latest version of a specific article. This functionality exists in the `GetDocument` method. Below is the complete code of that method, it demonstrates the recommended use of `SqlConnection` and the use of stored procedures with output parameters.

```
protected string GetDocument(
    int siteId,
    string path,
    string suffix)
{
    using (SqlConnection connection = new SqlConnection(_connectionString)) {
        SqlCommand command = connection.CreateCommand();
        command.CommandType = CommandType.StoredProcedure;
        command.CommandText = "GetDocument";
        command.Parameters.Add("@SiteId", SqlDbType.Int).Value = siteId;
        command.Parameters.Add("@Path", SqlDbType.NVarChar).Value = path;
        command.Parameters.Add("@Suffix", SqlDbType.NVarChar).Value = suffix;
        SqlParameter parameterDocument =
            command.Parameters.Add("@Document", SqlDbType.Xml);
        parameterDocument.Direction = ParameterDirection.Output;
        SqlParameter parameterIsCompiled =
            command.Parameters.Add("@IsCompiled", SqlDbType.Bit);
        parameterIsCompiled.Direction = ParameterDirection.Output;

        connection.Open();
        command.ExecuteNonQuery();

        string document = parameterDocument.Value is DBNull ?
            null : (string)parameterDocument.Value;

        if (document != null &&
            ((bool)parameterIsCompiled.Value) == true) {
            return document;
        }
    }
}
```

```

    else if (document != null) {
        return UpdateCacheDocument(document);
    }
    else {
        return null;
    }
}
}
}

```

The method is very simple since some of the logic resides in the stored procedure. The returned document should in most cases be a compiled document that is directly sent to the next step in the pipeline.

7.4.5 InterceptorHandler

Most wiki engines has support for dynamic content, this can either be as simple as returning the current date or a complex function that might use a web service to get stock quotes. KuaiWiki enables dynamic content by using a delegate called `InterceptorHandler`. Delegates of this type is added to an `Interceptor` class and associated with a fully qualified XML name. The wiki XML document parser checks if there is any associated `InterceptorHandler` for each element name parsed. If there is an associated `InterceptorHandler`, the delegate is invoked, the parameters to the delegate are the current `XmlReader` object, `Request` object and a `DateTime` output parameter.

```

public delegate object InterceptorHandler(
    XmlReader xmlReader,
    Request request,
    out DateTime expiryDate);

```

The method that the delegate invokes processes the content of the XML element and returns a result that is either an `XmlReader` or a `string`. The result is then replacing the original element. The result can contain nested dynamic elements to an arbitrary depth. The `expiryDate` parameter is used to control the amount of time that the dynamic result is valid and is used in the caching algorithm.

7.4.6 InterceptingXmlReader

The `InterceptingXmlReader` class is used to parse the XML into an `XPathDocument`. The `XPathDocument` is the preferred class to use as input to XSL transformations.

The `InterceptingXmlReader` was one of the hardest classes to implement. There are different ways to intercept dynamic elements. The simplest approach is to load the XML into the `XmlDocument` class, which provide a W3C XML DOM implementation. The `SelectNodes` method can be used to select elements based on an XPath expression. For each dynamic element name `SelectNodes` is used to get those elements, and then they are replaced with the result from the delegate. This is a very quick and easy way to implement the requested feature, but it has several drawbacks. For each dynamic element name, there has more or less to be one `SelectNodes` call containing an XPath expression of the form

`//element-name` where `element-name` is the name of the element to search for. This requires the `XmlDocument` to walk through all the nodes in the document and see if any of them have the specified name. This is a slow operation and is required for all element names, thus the performance would be dependent on the number of dynamic element names that is supported.

The `InterceptingXmlReader` uses a different approach. The class implements the `XmlReader` abstract class. This class contains over 80 methods and properties, and some of them are abstract. The `XPathDocument` class was studied, with the help of Lutz Roeder's reflector tool, and the conclusion was that only a few of the 80 methods are actually used by `XPathDocument`. Since `InterceptingXmlReader` was only designed to be used together with `XPathDocument`, only the limited set of methods were implemented.

`InterceptingXmlReader` is a wrapper around a stack of `XmlTextReader` classes. The XML string or stream returned by the `PageSource` object is feed into an initial `XmlTextReader` object that to begin with is the only reader in the stack. When the `Read` method is called, the next node of the `XmlReader` on top of the stack is checked to see if it is a dynamic element, if it is the related delegate is invoked. If the returned result is a string, it is returned as a text node, if the returned result is an `XmlReader`, it is pushed onto the stack. The following `Read` method calls will use this `XmlReader`. When the `Read` method reaches the end of an `XmlReader`, that `XmlReader` is removed from the stack.

This might seems like an easy implementation, but there are many properties and special cases that need to be handled.

Possible improvements

Microsoft released new documentation of the `XmlReader` and related classes during the development of KuaiWiki. The documentation contains some new recommendations, and describes some features that were not known at the time of the implementation of `InterceptingXmlReader`.

The new recommendations and functions should be studied to see if it possible to enhance the performance or simplify the implementation.

The `XmlReader` uses a `NameTable` object to cache attribute and element names. The new documentation contains some memory related warnings to this class that has to be considered.

It might be possible to use the `NameTable` class to lookup elements that are dynamic which is something that could enhance the performance. (MSDN: Reading XML with the `XmlReader`, 2005-11-17)

7.4.7 XPathNavigableConverter

The last step of the pipeline is to convert the `XPathDocument` into XHTML by using XSL stylesheets. Loading an XSL file and create a `XslCompiledTransform` is an expensive operation. Thus the `XslCompiledTransform` should be cached. In a wiki hosting scenario, it is very likely that many wiki webs will use the same skin, and the cached `XslCompiledTransform` should be shared between different `WebConfig` objects. One skin can exist of multiple XSL stylesheets that are combined into one.

The `StandardXslFactory` implements this caching functionality, and make sure that each skin is only instantiated once and placed in the cache.

An XSL transform can take a collection of variables that can be accessed from the stylesheet to modify the output. Some parts of stylesheets should be able to be modified with variables to avoid changes to the stylesheet. The variables are specified in the configuration file related to the specific wiki web.

Example of section from the configuration file:

```
<skin name="Strict">
  <xsl href="Strict.xslt">
    <variable name="copyright">
      Mattias Beermann <date/>
    </variable>
  </xsl>
  <xsl href="Base.xslt"/>
</skin>
```

The above section defines a skin called Strict. The skin consists of two XSL stylesheets that should be combined, Strict.xslt and Base.xslt. The Strict.xslt stylesheet defines one variable named copyright, it should be set to the value of Mattias Beermann and the current date. The `<date/>` element has to be replaced with the current date. The same functionality as used when parsing the wiki XML documents is used to replace the element with its dynamic value. The result is cached based on the ExpiryDate, just as with wiki XML documents.

Possible improvements

The `xPathNavigableConverter` implementation does not cache the transforms in an optimal way. The implementation can be improved, and could also contain functions related to CSS stylesheets.

There is no logic that defines which stylesheet should be used, the KuaiWiki engine should choose the appropriate stylesheet based on the browser.

7.5 Database

The database has been designed for optimal performance and to be able to scale. All data is saved in UNICODE, which are 16-bits compared to the normal 8-bits required for each character. This doubles the storage requirements for most languages, but has the major benefit of being able to handle all languages with the same codepage.

The database design is still young and future work would be to test the performance with a large number of documents to detect bottlenecks and poor design.

7.5.1 Table layout

Three tables are used:

Archive.Document - contains all documents in the wiki. An XML wiki document contains all required information about the document. The table columns SiteId, DocumentId and Version contain extracted data from the wiki document. The reason for extracting these XML values into columns

is to enhance query performance, since most queries to this table will be to request a certain version of an article that belongs to a SiteId. A clustered index is defined on these columns. The DocumentId is of the type uniqueidentifier, that is the same as a GUID, a global unique 16 byte id. Using an int datatype would create a denser index and thereby better query performance, but the GUID simplifies importation of documents from another wiki, since they are globally unique. The uniqueidentifier is also better to use when replicating values between different servers.

Cache.Document - this table contains a cached compiled version of the latest version of all wiki documents in the archive. Some of the dynamic information in the document have been processed and saved, thus there is no need to reprocess this information until it expires. This should result in better performance, especially for dynamic content that is expensive to get and that can be cached for a long duration. The table contains four columns, DocumentId, Document, ExpiryDate and CachedDate.

The DocumentId column is the clustered primary key.

The most frequently accessed version of a document is the latest version. These reside in the relative small table Cache.Document compared to the much larger Archive.Document table that contains all versions of the documents.

The smaller size of Cache.Document enables smaller indexes and since it is so frequently accessed, most of the table will reside in the memory cache on the SQL Server, thus enabling better query performance.

Cache.Path - this table contains mappings between a DocumentId and the related suffix, path, alias and site. The suffix is used to group documents into different namespaces. The articles reside in one namespace, and the discussion pages in another.

There is a clustered primary key on the columns SiteId, Path and Suffix.

Possible improvements

The indexes on the tables are carefully chosen, but should be tested on a large wiki site using the query optimization features in Microsoft SQL Server 2005. There is a possibility that the clustered indexes might create performance problems when inserting documents since the rows in the database are physically ordered by the clustered index.

The table layout should be tested and verified in a replicated environment, ensuring that the table layout works in most replication scenarios.

There are some functions that access individual elements inside the XML documents, these tasks can benefit from XML indexes, and it would be interesting to test the performance of XML indexes compared to extracting the values into a column or another table.

7.5.2 Stored Procedures

Stored procedures are used for all access to the database tables. The stored procedures adds one extra layer of flexibility, they can be changed without having to do any changes to the C# code of KuaiWiki.

The stored procedures are designed to minimize the number of calls between the KuaiWiki server and the database. For article requests, it is mostly only one database call that has to be made. When an article is inserted, extraction of *what links here* data and other such tasks are or should be performed in the database. If this was performed in the KuaiWiki engine, several calls have to be made to the database server, creating a larger load on the database server.

Only two of the many stored procedures are described in this section.

The most straightforward method is to select the compiled document, return it to the KuaiWiki engine, check if it has expired and if it has, call a new procedure that selects the archived version of the document. This would require two database calls.

The GetDocument stored procedure is constructed to return the compiled document if it has not expired, otherwise the uncompiled version is returned. By moving the logic into the stored procedure, one database call is eliminated, resulting in better network and database server utilization .

```
CREATE PROCEDURE GetDocument
    @SiteId int,
    @Path nvarchar(200),
    @Suffix nvarchar(20),
    @Document XML OUT,
    @IsCompiled bit OUT
AS
    DECLARE @DocumentId uniqueidentifier

    -- Try to get the compiled document.
    SELECT
        @Document =
            CASE
                WHEN ExpiryDate > GETDATE() THEN Document
                ELSE NULL
            END,
        @DocumentId = Cache.Document.DocumentId
    FROM
        Cache.Document
        INNER JOIN Cache.Path ON Cache.Path.DocumentId =
            Cache.Document.DocumentId
    WHERE
        Cache.Path.SiteId = @SiteId
        AND
        Cache.Path.Path = @Path
        AND
        Cache.Path.Suffix = @Suffix

    IF @Document IS NOT NULL
    BEGIN
        SET @IsCompiled = 1
        RETURN
    END
```

```

-- The compiled document has expired or there was no match.
SET @IsCompiled = 0

IF NOT @DocumentId IS NULL
BEGIN
    -- There was at least a match. Use the Guid to get the
    -- latest uncompiled version of the document.
    SELECT TOP 1
        @Document = Document
    FROM
        Archive.Document
    WHERE
        DocumentId = @DocumentId
    ORDER BY
        Version DESC

    IF @Document IS NOT NULL
    BEGIN
        RETURN
    END
END

```

The InsertDocument stored procedure is another example of moving some of the business logic into the stored procedure, and thereby avoiding several database calls. The steps performed are:

1. Calculate the version of the new document based on the documents in the archive with the same document id.
2. Insert the document into the archive.
3. Insert or update the cached version of the document.

```

CREATE PROCEDURE InsertDocument
    @SiteId int,
    @Document xml,
    @CachedDocument xml,
    @ExpiryDate datetime
AS
    DECLARE @DocumentId uniqueidentifier
    DECLARE @Version int

    -- Get the DocumentId
    SET @DocumentId = @Document.value(
        '/kuai[1]/document[1]/@id', 'uniqueidentifier')

    -- Get the Version
    SELECT
        @Version = MAX(Version) + 1
    FROM
        Archive.Document

```



```

WHERE
    DocumentId = @DocumentId

-- Insert the Document into archive
INSERT Archive.Document
VALUES (@SiteId, @DocumentId,
ISNULL(@Version, 1), @Document)

-- Insert/update the cached document into the cache
IF EXISTS(SELECT * FROM Cache.Document
WHERE DocumentId = @DocumentId)
BEGIN
    UPDATE Cache.Document
    SET Document = @Document, ExpiryDate = @ExpiryDate,
        CachedDate = GETDATE()
    WHERE DocumentId = @DocumentId
END
ELSE
BEGIN
    INSERT Cache.Document
    VALUES (@DocumentId, @Document,
        @ExpiryDate, GETDATE())
END
END

```

7.5.3 Triggers

Triggers can also be used to move some logic into the database. The trigger `Cache.PathAndAliases` updates the `Path` table when a cached document is inserted, updated or deleted.

The steps performed are:

1. Delete old paths and aliases.
2. Insert the paths that the new documents contained.
3. A document can have multiple aliases, for example an article about Beijing, could have the path `China/Beijing` and the aliases `China/Peking` and `China/Beijing` written using Chinese characters. The aliases are extracted from the document, and inserted into the `Path` table with the help of the XML features in Microsoft SQL Server 2005.

```

CREATE TRIGGER Cache.PathAndAliases
ON Cache.Document AFTER INSERT, UPDATE, DELETE
AS
-- Delete old paths and aliases
DELETE Path FROM Path INNER JOIN
    (SELECT DocumentId FROM inserted UNION
    SELECT DocumentId FROM deleted) AS Changed
ON Changed.DocumentId = Path.DocumentId

-- Insert new paths

```

```

INSERT Path SELECT
    (SELECT TOP 1 Archive.Document.SiteId
     FROM Archive.Document
     WHERE Archive.Document.DocumentId =
           inserted.DocumentId),
inserted.Document.value(
    '/kuai[1]/@path', 'nvarchar(max)'),
inserted.Document.value(
    '/kuai[1]/@path-suffix', 'nvarchar(max)'),
inserted.DocumentId,
0
FROM
    inserted

-- Insert new aliases
INSERT Path SELECT
    (SELECT TOP 1 Archive.Document.SiteId
     FROM Archive.Document
     WHERE Archive.Document.DocumentId = inserted.DocumentId),
alias.value('.', 'nvarchar(max)'),
inserted.Document.value(
    '/kuai[1]/@path-suffix', 'nvarchar(max)'),
inserted.DocumentId,
1
FROM
    inserted CROSS APPLY
        inserted.Document.nodes('/kuai/aliases/alias/@path')
        AS T(alias)

```

When a document is inserted, updated or removed in the Cache.Document table the *what links here* section of the document should be updated, and all documents that are linked from the new document should also be updated. This is performed with a trigger, one more example of how to avoid a lot of database calls between the KuaiWiki engine and the database server, resulting in better performance.

The steps performed are:

1. The inserted and updated documents have their *what links here* section updated.
2. The documents inserted, updated and deleted, contains links to other pages. These pages are modified to include an up to date *what links here* section.

```

CREATE TRIGGER Cache.WhatLinksHere ON Cache.Document
FOR INSERT, UPDATE, DELETE
AS
-- Create WhatLinksHere for the inserted documents
UPDATE Cache.Document
SET
    Cache.Document.Document = dbo.UpdateWhatLinksHere(

```

```

        Cache.Document.Document,
        Cache.Document.DocumentId)
FROM
    Cache.Document
    INNER JOIN inserted ON
        inserted.DocumentId = Cache.Document.DocumentId

-- Update WhatLinksHere for documents that are linked to
-- from the inserted or deleted documents.
UPDATE Cache.Document
SET
    Cache.Document.Document = dbo.UpdateWhatLinksHere(
        Cache.Document.Document,
        Cache.Document.DocumentId)
FROM
    Cache.Document
    INNER JOIN Cache.Path ON
        Cache.Path.DocumentId = Cache.Document.DocumentId
    INNER JOIN
        ( -- Select all hrefs in the modified documents.
        SELECT insertedHref.value('.', 'nvarchar(max)')
          AS Path FROM inserted CROSS APPLY
            Document.nodes('/kuai/document//@href')
          AS T(insertedHref)
        UNION
        SELECT deletedHref.value('.', 'nvarchar(max)')
          AS Path FROM deleted CROSS APPLY
            Document.nodes('/kuai/document//@href')
          AS T(deletedHref)
        ) AS Href ON Href.Path = Path.Path
GO

```

Possible improvements

Both these triggers recreate all the data even if it has not changed. Increased performance might be achieved by first testing to see if the data needs to be recreated, and only then recreate it.

The performance of the various XML query functions has not been evaluated. It would be interesting to test the performance difference between different implementations, both with using logic at the database server and in the KuaiWiki engine.

7.5.4 Functions

In addition to stored procedures and triggers, it is also possible to define functions. The Cache.WhatLinksHere trigger uses a function to update the *what links here* section in the documents.

The steps performed are:

1. Delete the existing what-links-here element. It is the parent element for all the links.

2. Create an empty what-links-here element.
3. Select and insert links to all Documents that has a link to @DocumentId

```

CREATE FUNCTION UpdateWhatLinksHere(
    @Document xml,
    @DocumentId uniqueidentifier)
    RETURNS xml
AS
BEGIN
    -- Delete old what-links-here element.
    SET @Document.modify('delete /kuai[1]/what-links-here')

    -- Create new what-links-here element.
    SET @Document.modify(
        'insert <what-links-here/> as last into /kuai[1]'
    )

    -- Select all Documents that has a link to @DocumentId
    DECLARE DocumentCursor CURSOR FAST_FORWARD FOR
    SELECT
        Document.value('kuai[1]/@path', 'nvarchar(max)')
    FROM
        Cache.Document CROSS APPLY
        Document.nodes('/kuai/document//@href') AS T(href)
    WHERE
        href.value('.', 'nvarchar(max)') IN
        (SELECT Cache.Path.Path FROM Cache.Path
        WHERE Cache.Path.DocumentId = @DocumentId)

    OPEN DocumentCursor

    -- A document containing a link to @Document
    DECLARE @Path nvarchar(max)
    FETCH NEXT FROM DocumentCursor INTO @Path
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @Document.modify('insert
            <link href="{sql:variable("@Path")}">
            {sql:variable("@Path")}</link> as last into
            (/kuai/what-links-here)[1]')
        FETCH NEXT FROM DocumentCursor INTO @Path
    END

    CLOSE DocumentCursor
    DEALLOCATE DocumentCursor

    RETURN @Document
END

```

The use of cursors should if possible be avoided due to performance reasons,

but in this case it seems unavoidable.

7.6 Future work

The KuaiWiki engine described here is designed for extensibility, performance and scalability. Best practices have been used in the design, and great care has been taken to ensure that no unnecessary processing takes place.

KuaiWiki is still in a pre-beta stage, and there are many features missing that are crucial for using it. Future work would be to implement the basic set of features required to make it work as a wiki engine. Then a large set of documents, preferably importing a database dump from Wikipedia should be used to test the performance and to tune the database design and the KuaiWiki implementation.

Once the KuaiWiki engine and the database layout is proven to work efficiently, all the caching features should be implemented, and the result before and after should be studied and evaluated. It should probably be possible to get web logs from Wikipedia and run the benchmarking based on real usage scenarios.

At this stage the KuaiWiki engine would be ready for a first beta release. If the implementation is proved to be stable, quick and usable, then the KuaiWiki code could be released as an open source project or sold as a commercial product.

Chapter 8

Conclusion

8.1 Document formats

An XML vocabulary for wiki articles could solve many of the problems of wiki text. The XML vocabulary presented in this thesis is the first step towards a standardized XML vocabulary. By using the ideas presented and with the help of wiki contributors and wiki implementers, a complete core wiki XML vocabulary could be defined. This XML vocabulary could be used to store and edit wiki content, and to be used as an interchange language between wiki engines using different wiki text dialects. The wiki XML vocabulary should be focused on structural markup, and not enable layout specific elements and attributes.

It is important to focus on the core of the vocabulary, and provide methods for extensibility and modularization. The same concepts used in the XHTML 2.0 draft recommendation could be used to achieve this.

The large number of existing well written and peer reviewed wiki articles is a valuable resource. Using a standardized markup language is one way to ensure that these articles can be read and easily presented in the future. The strict validation and well formedness provided by XML is crucial to ensure easy migration of articles from one wiki engine to another.

Wiki articles can not currently be easily transferred from one wiki text dialect to another. By standardizing on one language, it becomes much easier to move content from one wiki engine to another, creating a more competitive wiki engine market.

8.2 WYSIWYG Editing

The current editing experience consists of HTML textarea editing of wiki text articles. This requires the author to learn the specific wiki dialect used by that wiki engine. Wiki text's most commonly used markup elements are easy to learn and quick to write, but some articles contains very complex markup that sometimes is a mix of wiki text, HTML and CSS.

Wiki editing should be possible for almost anyone with the help of a browser based WYSIWYG XML editor. The editor should be aware of the DTD/Schema

of the wiki XML vocabulary, and only allow valid markup. The editing experience will be similar to using a word processor application.

Many problems need to be solved before a good WYSIWYG XML editor is possible. The limited set of features provided by the browsers, make most operations hard to implement. A simple concept such as *cut and paste* becomes a major problem due to the missing OnPaste event handler in Mozilla Firefox.

The developed experimental prototype highlighted some of these problems, but also presented some workarounds.

Schema aware XML editors are uncommon, and creating one that is browser based is not a task that should be underestimated, but it should be possible.

The design and architecture chapter described how AJAX technology could be used to provide a *wiki HTML application* implemented in JavaScript and DHTML, that enables page navigation that only requests the new article as an XML document from the server. The XSL transformation take place in the browser and only the article content area need to be updated. The user should be able to switch from view mode to editing mode without having to make a request to the server. By using AJAX in the described way, some of the processing is moved from the wiki engine to the client, thus each request will require less server processing time, enabling a larger number of requests per second and at the same time provide a better user experience.

8.3 Wiki engine implementation

The wiki engine implementation, code named KuaiWiki, was used to test some of the ideas presented in this thesis. The implementation was designed to use XML, XSL transformations, and to be modular and have high performance and scalability.

The use of a modified version of Microsoft's provider pattern enables the wiki engine to become a framework that can be used to explore different solutions. Most parts of KuaiWiki can be replaced by other classes by making changes to the configuration files. This should be very valuable when several different implementations exist for one specific problem. By creating two configurations that share everything except that part, it should be easy to conduct performance benchmarking.

KuaiWiki was designed to be lightweight, to be able to run as a small stand-alone server application, and to scale up to wiki hosting scenarios. Most wiki engines requires a separate installation for each wiki web, the wiki engines that have support for wiki hosting has often added this feature afterwards, creating none optimal solutions. KuaiWiki was designed from the beginning to support multiple wiki webs running on the same instance and supports specific configurations for each web, even parts of the wiki engine can be changed due to the modified provider pattern.

The XML features of Microsoft SQL Server 2005 were explored and it was proved that some of the logic can be moved from the wiki engine to the database. There is a balance between what should be done in the wiki engine and in the database. Features that require little processing and a lot of data access should reside in the database, features that require a lot of processing should reside in the wiki engine. This is to ensure that one database server can serve many wiki engines. This simplifies a scale out scenario were a two machine setup is

not enough, due to the fact that it is easier to scale out the wiki engine than to scale out the database.

The `InterceptingXmlReader` class enables elements to be intercepted and replaced with dynamic data. The implementation demonstrated that it is possible to only implement the subset of methods and properties of the abstract `XmlReader` class that the `XPathDocument` class uses. By intercepting the elements at parse time, a costly XML DOM solution can be avoided, resulting in better performance and less memory requirements.

The implementation proved that it is much harder to architect, design and write a program that is optimized for performance and that enables the application to scale out and up. The different caching features create a design that is much more complex than without caching.

The .NET Framework documentation often lacks information about performance. By using Lutz Roeder's Reflector tool it was possible to examine how the classes in the .NET Framework are implemented, and thereby the classes could be used in a more optimized way.

The KuaiWiki implementation could be used to make further research about how wiki engines should be designed, and how they can use XML and XSL transformations. When the implementation has matured, it should go through extensive benchmarking and profiling to optimize the implementation further.

8.4 The wiki concept

The wiki concept has existed for ten years, during this time there have been many success stories. Wikipedia.org is one of them, a wiki web containing more than one million articles, in a large number of languages. It provides an up to date and peer reviewed encyclopedia that every person with internet access can use and contribute to free of charge. Wikipedia and its sister projects are growing faster and faster, and if spam and vandalism can be controlled, they will probably continue to grow for a long time to come.

The simplicity of the wiki concept enables anyone to be a contributor. There are many wikis specialized on one topic and as wiki hosting becomes more common and better, wiki sites might be as numerous as Blogging sites. A wiki site can be used as an online collaborative management system, as the functionality of wikis increase, they will probably be an alternative to traditional collaborative management systems. Wiki sites are becoming a tool for companies to use on their intranets to manage information.

Wiki sites will probably get better support for printing, enabling the user to buy a printed book with a set of wiki articles. This could be very useful for the Wikibooks projects. The accessibility features will probably improve, and the improved text-to-speech technology could be used to enable users to download audio-files of articles.

A simple and naive approach to create valuable content online has been proved to work. The wiki concept will continue to evolve and this thesis has provided some possible directions.

Bibliography

- Alexa - Global Top 500. 2005-11-05.
http://www.alexa.com/site/ds/top_sites?ts_mode=global&lang=none.
- Amazon: Web services. 2005-11-06.
<http://www.amazon.com/gp/browse.html/103-9359605-9669417?node=3435361>.
- Apache: FOP. 2005-11-06.
<http://xmlgraphics.apache.org/fop/>.
- Lars Aronsson. Operation of a large scale, general purpose wiki website. 2002-11-07.
<http://aronsson.se/wikipaper.html>.
- DotGNU. 2005-11-06.
<http://en.wikipedia.org/wiki/DotGNU>.
- Sergey Dubinets and Anton Lapunov. Migrating to xslcompiledtransform. 2005-11-07.
<http://blogs.msdn.com/xmlteam/archive/2005/09/30/475922.aspx>.
- FlexWiki: FlexWiki Features. 2005-11-11.
<http://www.flexwiki.com/default.aspx/FlexWiki.FlexWikiFeatures>.
- FlexWiki: Wiki Federation Overview. 2005-11-12a.
<http://www.flexwiki.com/default.aspx/FlexWiki/WikiFederationOverview.html>.
- FlexWiki: Wiki Federation Overview. 2005-11-12b.
<http://ww.flexwiki.com/default.aspx/FlexWiki/WikiPageProperty.html>.
- Rob Howard. Provider model design pattern and specification, part 1. 2005-11-07.
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspnet/html/asp02182004.asp>.
- htmlArea. 2005-11-15.
<http://www.htmlarea.com/>.
- Craig Larman. *Applying UML and patterns*. Prentice Hall PTR, 2002. ISBN 0-13-092569-1.
- Bo Leuf and Ward Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley Professional, 2001. ISBN 020171499X.

Lutz Roeder. 2005-11-06.
<http://www.aisto.com/roeder/dotnet/>.

MediaWiki - Documentation: Introduction. 2005-11-11.
<http://meta.wikimedia.org/wiki/Documentation:Introduction>.

J.D. Meier, Srinath Vasireddy, Ashish Babbar, and Alex Mackman. *Improving .NET Application Performance and Scalability*. Microsoft Press, 2004. ISBN 0735618518.
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenet.asp>.

Microsoft SQL Server 2005 Product Overview. 2005-11-06.
<http://www.microsoft.com/sql/2005/productinfo/overview.msp>.

Mono. 2005-11-06.
<http://www.mono-project.com>.

MSDN: Comparing XML Performance. 2005-11-06.
<http://msdn.microsoft.com/vstudio/java/compare/xmlperf/default.aspx>.

MSDN: Reading XML with the XmlReader. 2005-11-17.
[http://msdn2.microsoft.com/en-us/library/9d83k261\(en-US,VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/9d83k261(en-US,VS.80).aspx).

Open Wiki: Xml Formatting. 2005-11-13.
<http://openwiki.com/ow.asp?OpenWiki>

David Ornstein. Flexwiki posted to sourceforge.net. 2005-11-11.
<http://blogs.msdn.com/dornstein/archive/2004/09/27/235042.aspx>.

Susning.nu: Homepage. 2005-11-08.
<http://susning.nu/>.

TPC.org. 2005-11-06.
<http://www.tpc.org>.

VeriTest. 2005-11-06.
<http://www.gotdotnet.com/team/compare/veritest.aspx>.

W3C: Extensible Markup Language 1.0. 2005-11-14.
<http://www.w3.org/TR/2004/REC-xml-20040204/>.

W3C: HTML Home Page. 2005-11-14.
<http://www.w3.org/MarkUp/>.

W3C: Modularization of XHTML 1.0 - Second Edition (Working draft). 2005-11-14.
<http://www.w3.org/TR/2004/WD-xhtml-modularization-20040218/>.

W3C: The Extensible Stylesheet Language Family. 2005-11-14.
<http://www.w3.org/Style/XSL/>.

W3C: XHTML 2.0 Draft. 2005-11-15.
<http://www.w3.org/TR/2005/WD-xhtml2-20050527/>.

W3C: XML Linking Language 1.0. 2005-11-15.
<http://www.w3.org/TR/xlink/>.

Wikimedia: About the Wikimedia Foundation. 2005-11-08.
http://wikimediafoundation.org/wiki/About_Wikimedia.

WikiMedia: Enable TeX. 2005-11-12.
http://meta.wikimedia.org/wiki/Enable_TeX.

Wikimedia: Home. 2005-11-08.
<http://wikimediafoundation.org/wiki/Home>.

WikiMedia: MediaWiki FAQ. 2005-11-12.
http://meta.wikimedia.org/wiki/MediaWiki_FAQ.

Wikimedia: Our projects. 2005-11-08.
http://wikimediafoundation.org/wiki/Our_projects.

Wikimedia servers. 2005-11-05.
http://meta.wikimedia.org/wiki/Wikimedia_servers.

Wikipedia: C Sharp. 2005-11-06.
http://en.wikipedia.org/wiki/C_Sharp_programming_language.

Wikipedia: DocBook. 2005-11-14.
<http://en.wikipedia.org/wiki/DocBook>.

Wikipedia: HTML. 2005-11-14.
<http://en.wikipedia.org/wiki/Html>.

Wikipedia: Java programming language. 2005-11-06.
http://en.wikipedia.org/wiki/Java_programming_language.

Wikipedia: Main page. 2005-11-13.
http://en.wikipedia.org/wiki/Main_Page.

Wikipedia: PHP. 2005-11-06.
<http://en.wikipedia.org/wiki/PHP>.

Wikipedia: Susning.nu. 2005-11-08.
<http://en.wikipedia.org/wiki/Susning.nu>.

Wikipedia: Technical FAQ. 2005-11-05.
http://en.wikipedia.org/wiki/Wikipedia:Technical_FAQ.

Wikipedia: Wiki. 2005-11-08.
<http://en.wikipedia.org/wiki/Wiki>.

Wikipedia: AJAX. 2005-11-05.
<http://en.wikipedia.org/wiki/AJAX>.

Wikipedia: Web service. 2005-11-06.
http://en.wikipedia.org/wiki/Web_service.

WikiWikiWeb: Moin Moin. 2005-11-11.
<http://c2.com/cgi/wiki?MoinMoin>.

WikiWikiWeb: TikiWiki. 2005-11-11.
<http://c2.com/cgi/wiki?TikiWiki>.

WikiWikiWeb: Top Ten Wiki Engines. 2005-11-11.
<http://c2.com/cgi/wiki?TopTenWikiEngines>.

WikiWikiWeb: Twiki Clone. 2005-11-11.
<http://c2.com/cgi/wiki?TwikiClone>.

WikiWikiWeb: WakkaWiki. 2005-11-11.
<http://c2.com/cgi/wiki?WakkaWiki>.

WikiWikiWeb: Wiki Engines. 2005-11-11.
<http://c2.com/cgi/wiki?WikiEngines>.