Fine-Grained Addressability to Support Large-Scale Collaborative Document Development

Viral Gupta
College of Computer and Information Science
Northeastern University
Boston, MA 02453, USA.
viral@ccs.neu.edu

Submitted to the College of Computer and Information Science in partial fulfillment of the requirements for the degree of Master of Science in Computer Science at the Northeastern University

©2008. Northeastern University. All rights reserved.

August 2008
Abstract

The Web has made it possible for large, distributed collaborations to develop sophisticated document bases. As these collaborations increase in size, there is a need to support reference, navigation and search through the document bases by individuals who are not computer professionals. Although collaborative tools such as wikis have been developed that address some of the requirements of these communities, the tools lack support for simple and convenient fine-grained addressability to parts of the documents. Such addressability is essential for formal documents, such as standards and legal documents. In this thesis we develop a solution to the problem of fine-grained addressability that is based on MediaWiki, a popular and powerful collaboration tool that is the software infrastructure for Wikipedia. Although some collaboration tools that support fine-grained addressability already exist, they have not addressed some of the open research issues of fine-grained addressability, such as dealing with transclusion, semantic annotation and hyperscope support. It also deals with the research problems that were raised by this goal. An architecture and reference implementation was developed to provide a proof of concept and to test the viability of the proposed solutions to the research problems. The thesis also discusses the various design decisions that were made in the course of solving the research problems and developing the reference implementation.
Acknowledgments

I am very grateful for the advice and support of my advisor, Professor Kenneth Paul Baclawski, for his feedback and for keeping me focused in my research. The comments and time given by my reader Professor Richard Kesner have greatly improved and clarified this work. I am thankful to Peter P. Yim and Jonathan Cheyer for the discussions which have helped a lot in capturing the user requirements properly. Finally I owe a lot to my colleague Tejas Parikh who has helped in developing some parts of the software. Our conversations and work together have greatly influenced this thesis.
List of Figures

2.1 Line numbers in a US patent ................................................. 4
2.2 Diagram labels in a US patent ............................................... 5
2.3 References to diagram labels in a US patent .............................. 6
2.4 HID numbers assigned to nodes in HTML document .................... 9
2.5 HID numbers assigned to nodes in HTML document .................... 10
2.6 NID identifiers assigned to nodes in HTML document .................. 11
2.7 NID identifiers assigned to nodes in HTML document .................. 12

3.1 As can be seen in the figure: document A, B and C are transcluded in document D ................................................................. 14
3.2 Transcluded nodes in Page 2 ................................................... 15
3.3 Transclusion can break if the structure of document changes .......... 16
3.4 HID numbers assigned to nodes in HTML document .................... 17
3.5 Node transclusion and Branch transclusion ............................... 17
3.6 Transclusion of HTML element p from page 1 to page 2 ............... 19

4.1 SYSTEM ARCHITECTURE ...................................................... 22
4.2 HID Numbering schemes ...................................................... 23
4.3 Transcluded Node on page 2 has presentation form compatible with the located in the page it is shown on ....................................... 25
4.4 Transcluded Node on page 2 has presentation form compatible with the located in the page it is shown on ....................................... 25

6.1 System Architecture ........................................................... 28
6.2 Relative Ranks of HTML Elements ......................................... 30
6.3 HID Algorithm ................................................................. 30
6.4 HIDs Assignment .............................................................. 31

8.1 Screen shot of wiki page showing HIDs assigned to html elements .... 34
8.2 Screen shot of wiki page showing HIDs assigned to html elements .... 35
8.3 Screen shot of wiki page showing transcluded nodes .................... 35
8.4 Screen shot of wiki page showing transcluded nodes .................... 36
Chapter 1

Introduction

The Internet and the World Wide Web have made it possible for large geographically distributed communities to share a large number of documents. Unfortunately, as the number and size of documents increases it becomes very difficult to navigate and to find information. The overall problem is to facilitate efficient and convenient creation, discussion and modification of these shared document bases by hundreds and thousands of participants in a community. Wikis have recently emerged as an effective tool for large scale collaboration and they have been successful for informal document development. The best known example is Wikipedia. However, wikis have serious shortcomings for formal document development such as standards, reports, legal documents, etc. The problem is that in the currently existing web infrastructure, HTML links are effective for linking to entire pages but very awkward for linking to parts of pages. Yet formal document development requires the ability to refer to small parts of a page: single paragraphs or even sentences. This capability is called fine-grained addressability. The ability to link to different sections in an HTML page is only possible if the author of the page had the foresight and technical background to create appropriate anchors for links. The process of creating these anchors requires some degree of computer sophistication which most of the members of the community would not have. We propose to develop an infrastructure based on wiki software that provides fine-grained addressability.

The most popular and powerful existing wiki software is MediaWiki, the engine behind the Wikipedia. While wikis have been developed that have the ability to link to sections of a document, these wikis are not very popular and do not have the powerful features of MediaWiki. MediaWiki, on the other hand, has only a very limited ability for fine-grained addressability. We propose to extend MediaWiki to support fine-grained addressability. We call this extension Purple MediaWiki Extension (PMWX). Because the MediaWiki infrastructure is complex, extending it as we propose involves solving a variety of research problems. If these anchors have to be created automatically by software then some questions need to be answered, such as up to what level of granularity should the elements in the page be addressable, and what elements in an HTML page should have automatically generated anchors. The solution must also deal with a number of significant usability and computer-human interaction (CHI) issues so that the extension will be usable by people who are not computer professionals.

Fine-grained addressability not only provides the ability to refer to parts of a document, it also provides the infrastructure for a number of other capabilities that are important for large scale collaborative development efforts, such as semantic annotation, transclusion, and hyperscope support.

Transclusion is defined as the inclusion of the content of all or part of one document into another document by reference. Once fine-grained addressability is achieved, document parts can be transcluded independently from one document into another. This leads to many other
research issues such as what sections of the page can be transcluded into other pages, can multiple sections of a given page be referenced by just referring to one section, how will the text from one page be presented on the other page, will the text have the presentation form of the page it is coming from or should it have a presentation form compatible with where it is located in the page it is shown on.

This thesis solves the problem of providing fine-grained addressability to support large-scale collaborative development of formal document bases by extending a popular and powerful collaboration infrastructure. It also deals with the research problems that were raised by this goal. An architecture and reference implementation for PMWX was developed to provide a proof of concept and to test the viability of the proposed solutions to the research problems. The thesis also discusses the various design decisions that were made in the course of solving the research problems and developing the reference implementation.
Chapter 2

Granular Addressability

Fine-grained addressability has a long history, going back many centuries. Perhaps the oldest examples are in religious texts such as the Torah, Bible and Qur’an which are indexed down to the level of individual verses. The Qur’an has an especially complex organizational structure, and there are several competing divisions into verses.

In this section some of many traditional uses of fine-grained addressability. We then give a brief history of the notion of Purple Number which is the basis for our introduction of fine-grained addressability into MediaWiki.

2.1 Traditional uses of fine-grained addressability

There are many domains that make use of fine-grained text and image addressability. In the legal domain, laws and regulations are labeled with hierarchical identifiers, and these identifiers become terms in their own right. For example, officially recognized tax-exempt charitable organizations in the United States are often called 501(c)(3) organizations even in non-legal contexts.

Another important example of fine-grained addressability is in patents. In the United States patents make use of line numbers for fine-grained references to the text of the document as seen in Figure 2.1. This excerpt and those in Figures 2.2 and 2.3 were taken from [1]

Figure 2.1: Line numbers in a US patent

The line numbers are locally unique on each page of the patent. In addition to fine-grained addressability of text, all drawings in US Patents must have every element of the drawing labeled
The identifiers for drawings are globally unique within the patent. The patent text refers to the elements of the drawings by using these identifiers as shown in Figure 2.3.

Scientific research papers, such as this one, use a mix of hierarchical and sequential identifiers. The precise style that is used depends on the scientific domain. While the text of the document is organized hierarchically, other items such as figures, equations, theorems and literature citations often use sequential numbering.

Government and corporate archives are increasingly being digitized and indexed. There are also several large-scale efforts by libraries to digitize books that are no longer protected by copyrights. The Encyclopedia of Life [2] is an example of such an effort. These efforts include both the original images of the documents and information extracted using OCR techniques. Fine-grained identifiers will be of increasing important to link extracted information with its source.

Standards commonly employ fine-grained addressability. This is especially important for the standards review process which can involve a large number of organizations and people. For example in the standards review process employed by UN/CEFACT [3], each line in the technical specification is given a line number. Reviewers use these line numbers to refer to the topic of interest in their reports and communications.

In general, when a document is illocutionary (i.e., performs a function beyond just being a narrative), then precise identifiers serve an important role.
The computer system **800** includes a central processing unit (CPU) **805**, which may include a conventional microprocessor, random access memory (RAM) **810** for temporary storage of information, and read only memory (ROM) **815** for permanent storage of information. A memory controller **820** is provided for controlling system RAM **810**. A bus controller **825** is provided for controlling bus **830**, and an interrupt controller **835** is used for receiving and processing various interrupt signals from the other system components.

---

**2.2 Granular Addressability idea when applied to web (HTML)**

Currently on the web is, HTML links are nearly always used to the entire page. They give very little about the exact location of the information in the entire page. To pin point exactly the content on a given page, links to different sections in a HTML page needs to be created by the author of the page. Creating links pointing to different sections within a page may work well for professional programmers and website designers, but it is difficult for ordinary users to include the required anchors. The naïve user on the web is assumed not to have any sophisticated skills and so the process of addressing different sections in a page needs to be automated.

**2.2.1 Automatically generate anchors for content in HTML**

If anchors for content within an HTML page need to be generated automatically what needs to be understood is, up to what level of granularity should the content within the HTML page be addressable. HTML has the data and presentation information embedded within. Some of the obvious choices can be: generate anchors for each HTML element within the HTML document, each word within the HTML document or each character within the HTML document.

Generating anchors for different HTML elements within a given HTML document will help the viewer go directly to it. This itself can be very powerful when a user is searching for a given text within a huge HTML document. So the immediate next thing which comes up is should anchors be generated to each and every HTML element occurring in the HTML document.

HTML has both the presentation and data within. It provides a means to describe the structure of text-based information in a document by denoting certain text as links, headings, paragraphs, lists, and so on and to supplement that text with **interactive forms**, embedded **images**, and other objects. There are different HTML tags like `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<p>`, `<li>`, `<table>`, `<img>`, `<td>`, `<tr>`, `<b>`, `<i>`, `<font>`. Tags `<h1>` - `<h6>` are called header tags. `<p>` is paragraph tag. `<li>` is list tag, `<table>` is table tag, `<b>` is bold tag which changes the text within the tag to bold, `<i>` is italics tag which changes the text within the tag to italics, `<font>` is used to change the font of the text within it. Tags `<b>`, `<i>` and `<font>` need not appear in isolation. They are be applied to an entire HTML element. For example:

```
<p> <b> example of bold text </b> </p>
<p> <i> example of italics text </i> </p>
<p> <font size=10> example of bold text </font> </p>
```

There is a difference between `<h1>` - `<h6>`, `<li>`, `<p>`, `<table>`, `<td>` and `<b>`, `<i>`, `<font>` tags from perspective of for which tags should have generated anchors. As can be
seen from the above examples, generating anchors for `<p>` tags ensures data within the html document is addressable so there doesn't seem any necessity to generate anchors for tags `<b>`, `<i>`, `<font>`. Thus there is no need to generate anchors for HTML tags which merely change the presentation of the HTML elements within the document. Thus it was necessary to identify the HTML tags for which elements will have generated anchors. After a lot of discussions the following HTML tags were selected: `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<p>`, `<td>`, `<table>`, `<img>`, `<li>`. Note each of the above mentioned HTML tag wraps within it some data of the page. Generating anchors for the above HTML tags makes different data elements within the HTML document uniquely addressable.

2.2.2 Different schemes for numbering HTML elements

To address different elements within HTML document we add anchor tags that look something like below.

```html
<a id="idnumber" href="#idnumber">(idnumber)</a>
```

The idnumber is the node identifier which is associated to a particular data element within a HTML document. The idnumber generation process can follow some rules so that it conveys some meaning or else can be generated in a total arbitrary fashion. The earlier work in this area have referred to idnumber as a Purple Number.

History of Purple Numbers

The concept of a Purple Number has its roots in the oNLine System (NLS) [4], which was a revolutionary collaboration system designed by Douglas Engelbart and his team in the Augmentation Research Center (ARC) at the Stanford Research Institute (now SRI International) during the 1960s and 1970s. NLS was the first to employ the practical use of hyperlinked documents (hyperdocs), the mouse (co-invented by Engelbart and colleague Bill English), raster-scan video monitors, information organized by relevance, screen windowing, computer presentation, and other modern computer concepts. The ARC team used NLS to collaborate in ways that are just now becoming available with today's Web 2.0 social networking software. NLS was subsequently renamed as the AUGMENT system when it was commercialized.

The first use of Purple Numbers on web pages can be traced back to the mid-1990s when Doug Engelbart along with Bob Czech and Christina Engelbart at the Bootstrap Institute [5][1][4], came up with the notion of placing Statement Numbers on page elements such as headers, paragraphs and figures. The intention was to provide Precision Browsing, mimicking the location number feature of the AUGMENT system. Christina, who developed the Bootstrap website, made those Statement Numbers purple in color, and as a result, Purple Numbers got its name. They were, however, just labels for reference purposes at the time.

Frode Hegland, who worked with Doug Engelbart around the time of the Bootstrap UnRev2 Colloquium at Stanford (Q1/2000), suggested some major enhancements to the earlier implementation of the web-based Purple Numbers. In particular, the Statement Numbers were made active. With that, purple numbers are associated with the link information of the anchor to the particular element (heading, paragraph, figure, etc.), and this capability is supported in all current implementations.

Today, there are two kinds of Purple Numbers. A hierarchical identifier (HID) is the current name for a Statement Number. HIDs are stateless and give hierarchical information about the document element. A node identifier (NID) or statement identifier is a unique identifier for a document element (or node) that is independent of the placement of the element within
the hierarchy of elements in the document. For a more detailed history of Purple Numbers see [1] and the links on this site. Encompassed in Doug Engelbarts bootstrap philosophy [5] is the notion of a networked improvement community collaborating to develop a collective intelligence by improving on an improvement infrastructure. In particular, a virtual community using a collaboration tool to develop and continuously improve on their collaboration and their collaboration tools is one instantiation of bootstrapping. Adhering to the bootstrap philosophy, this paper, as well as the PMWX project, has been using a PMWX-enabled MediaWiki site both for the project [6] and the writing of the paper [7].

Purple Numbers are currently being used successfully in research, academia, government and commercial settings.

One such deployment is in the Ontolog communities Collaborative Work Environment hosted on the CIM3 infrastructure. Ontolog (a.k.a. Ontolog Forum) is an open, international, virtual community of practice devoted to advancing the field of ontology, ontological engineering and semantic technology, and advocating their adoption into mainstream applications and international standards [8]. Peter Yim, is the founder of CIM3, as well as one of the founders of the Ontolog Forum.

There are a number of examples of very successful projects in (the US) government that made use of Purple Numbers. Susan Turnbull of the GSA Office of Intergovernmental Solutions has conducted a series of Collaborative Expedition Workshops with multiple Communities of Practice to advance government-to-government and government-to-citizen collaboration [9]. Another example was in their use to augment the development of the (US) Federal Enterprise Architecture, Data Reference Model v2.0 standard [10]. That development activity involved more than 300 documents, 585 people in 8 teams, and 5 workshops. This standard had an impact on virtually every US government agency. The new standard was developed in just 6 months, a pace that is rarely achieved in a standards development activity. In both of these projects, Purple Numbers served as a mechanism for rapidly organizing a very complex series of discussions and negotiations.

Hierarchical Identifiers

Hierarchical Identifiers (HID) are Purple Numbers that represent a way to identify nodes in a structured document. As the name implies, each node in a structured document can be classified by its hierarchical position in the document. HIDs correspond to the “statement numbers” in the NLS system.

The central idea is that the web document can be seen as a tree wherein each HTML element within it is a child of root element. Thus an HTML element can have a sibling relationship, child/parent relationship with other node in the entire tree. An addressing scheme which treats a web document as a tree is the most intuitively understandable way to view the web document.

If the web (HTML) document keeps changing quite often the hierarchy of the elements within the document may change. Consider the Figure 2.4.

In this the hierarchy of element \textless h2\textgreater changes in state 2. Note here the hierarchy of nodes change due to the introduction of a new node in the document.

As can be seen in state 1: element h1 has HID 1; p element has HID 1A indicating it is child of element h1; h2 element has HID 1B indicating it is child of element h1 and sibling of element p. As can be seen in state 2: element h1 has HID 1; p element has HID 1A indicating it is child of element h1; h3 element has HID 1B indicating it is child of element h1 and sibling of element p; h2 element has HID 1C indicating it is child of element h1 and sibling of element p, h2.

Also consider the Figure 2.5 where the hierarchy of the node \textless h2\textgreater changes due to the
Figure 2.4: HID numbers assigned to nodes in HTML document
deletion of a node within the document. Thus for non static documents the hierarchy keeps changing after each version and so the HIDs need to be generated again.

Node Identifiers (NID) are Purple Numbers very similar to the “statement identifiers” of the NLS system. An NID goes beyond specifying a hierarchical location to a nodes or furnishing an anchor for the node. The key requirement of a Node Identifier is that it must be stateful. Once an NID is assigned to particular node in the document the NID remains with that node for the lifetime of the node. To ensure this property, NIDs are stored in a database together with the document to preserve their state.
The key distinguishing point between HIDs and NIDs is that an HID specifies the hierarchical location of a node in a document whereas an NID is assigned to the node itself. Note the NID identifier assigned to a node does not give any information about the hierarchy of the node in the given document. It is a plain number which does not convey any meaning.

It is in fact difficult for NIDs to reflect hierarchy information, as the content on a wiki keeps changing and so the positions of the nodes in a given web document, while the node which is once assigned an identifier cannot be reassigned.

![Figure: State 1 of HTML document](image1)

![Figure: State 2 of HTML document](image2)

Figure 2.6: NID identifiers assigned to nodes in HTML document

In Figure 2.6 in state 2 when element h3 is added it is assigned a new id, thus maintaining the previous id's which were assigned. Thus as the document keeps changing it is difficult to understand the hierarchical position of a node just by looking at its NID.

Each time the wiki page is edited and saved, the content of the wiki page is scanned for new nodes. Each new node gets assigned a new unique NID which will identify that particular node only. An interesting point here is when a node is edited, e.g., a line is added in middle of
a paragraph which already has an assigned NID, it should not be considered to be a new node even though the content in that node has changed. If a user wishes to assign a new NID to a node whose content is changed, the user must delete the NID assigned to that node. A new NID will be assigned when the page is saved. In particular, when a user deletes a node, the NID for the node should also be deleted.

Figure 2.7: NID identifiers assigned to nodes in HTML document

In the Figure 2.7, NID of node h2 remains the same even after the content of h2 element changes.
Chapter 3

Transclusion

Transclusion is defined as the inclusion of the content of a document into another document by reference. For example, when someone is writing a document on a particular country and needs to pull information about the cities from different documents.

3.1 The current state of Transclusion on web

Transclusion in HTML is somewhat limited by lack of standards support in Internet Explorer. Although all graphical browsers can transclude an image, including a document is a bit more difficult. There are currently two methods of achieving this result:

- the Iframe (inline frame) element which includes the whole document.
- The Object element. Firefox and other browsers allow true document transclusion, enabling one page to be built from several smaller documents.

In addition, Ajax can achieve a similar result across all modern, JavaScript-enabled browsers. Future versions of HTML may support deeper transclusion of portions of documents using XML technologies such as entities, XPointer document referencing, and XSLT manipulations.

3.2 Transclusion using the power of granular addressability

The way transclusion works on the web is as shown in the Figure 3.1. The document D has references to documents A, B and C. The transcluded text could be pulled since the documents A, B and C were uniquely addressable on the web.

With granular addressability each HTML element within the HTML document on the web has a unique address (complete URL). There cannot be two HTML elements with the same URL. Thus even sections within an HTML document can be transcluded using the power of granular addressability. This leads to very interesting scenarios. If the only important piece of information existing on a given HTML page can be transcluded leaving aside other content, then it can avoid all the clutter.

As discussed in section 2.2.2 there are different numbering schemes for addressing. These are called HID (Hierarchical Identifiers) and NID (Node Identifiers). Using each of these addressing schemes gives user different capabilities.
Figure 3.1: As can be seen in the figure: document A, B and C are transcluded in document D
3.2.1 Transclusion using HID numbering scheme

When the user transcludes text using the HID numbering scheme, the user needs to know the
HID of the element within the page which has to be transcluded. As shown in the Figure 3.2,
element p from page 1 is transcluded into page 2. This is indicated by a mark T by the side
of the element. Similarly element h1 on page 1 can be referred to as page1#1. Thus as shown
individual HTML elements on the page can be transcluded easily using HIDs.

![Figure 3.2: Transcluded nodes in Page 2](image)

Figure 3.2: Transcluded nodes in Page 2

If HID of the node which needs to be transcluded from page 1 needs to be mentioned on
page 2, the transclusion will break if page 1 changes its structure. As the document changes
and so the structure of the document also changes, HIDs assigned to html elements depend on
the position of the node in the html tree and so HIDs are reassigned in each new version of the
As shown in the Figure 3.3, the user while specifying HID 1A on page 2 expects always html element p to be transcluded. But once the structure of page 1 changes a different HID is assigned to element p and so the same element is not transcluded. Thus transclusion can very easily break when using HIDs to refer to html elements within documents.

Figure 3.3: Transclusion can break if the structure of document changes

Transclusion can be used in very powerful ways when using HIDs because of the way numbers are assigned to HTML elements.

Consider the document in the Figure 3.4. The document is capturing meeting notes and there are different parts in it, attendees, names of the attendees, discussion points, issues to be discussed. Clearly the HTML elements have related information. If a user wants to transclude the part in italics into page 2. The user can write transclude tag for each element separately or else can simply transclude the branch element 1A. By doing so element p and all its child elements are transcluded as shown in the Figure 3.4.
Figure 3.4: HID numbers assigned to nodes in HTML document

Figure 3.5: Node transclusion and Branch transclusion
Thus instead of transcluding individual nodes separately the entire branch can be transcluded. Similarly if the user had to transclude the HID 1B, 1B1 and 1B2 the following could be used.

\[
\text{<tb pageid=page1 nodeid=hid1B> </tb>}
\]

This power to transclude the entire branch comes from the way the HID numbering is assigned to elements.

3.2.2 Transclusion using NID numbering scheme

When the user transcludes text using the NID numbering scheme, the user needs to know the NID of the element within the page which has to be transcluded. As shown in Figure 3.6, element p from page 1 is transcluded into page 2. This is indicated by a mark T by the side of the element. Similarly element h1 on page 1 can be referred to as nodeid=nid1. Thus as shown individual HTML elements on the page can be transcluded easily using NIDs.

Note in the Figure 3.6, NID values can be anything they do not follow the hierarchical numbering pattern.

Unlike HIDs, NIDs are stateful and once assigned remain with the node for the lifetime of the node. When using NIDs the transclusion cannot break as once a given node in HTML document has an nid assigned it remains with it.
Figure 3.6: Transclusion of HTML element p from page 1 to page 2

Figure: element p of page 1 transcluded on page 2
Chapter 4

Purple Mediawiki (PMWX)

Purple Mediawiki is software developed for Mediawiki, the source code for Wikipedia, one of the most widely used wikis. The above discussed ideas of granular addressability and transclusion have been implemented for Mediawiki. The development effort was undertaken by following all the phases of the software development life cycle.

4.1 Requirements and Specifications

This section throws more light on the key features of the PMWX system i.e. Purple Numbers and transclusion support.

4.1.1 Hierarchical Identifiers

The idea of an HID (Hierarchical Identifier) is given in the section 2.2.2. One of the main requirement of HIDs is they must be stateless. A Purple Number is assigned to the hierarchy of the node and not the node itself. An HID is assigned to every node in the document that has a unique parent node. The second requirement is that HIDs should be unique for any given page. This is important as this makes sure every HTML element is uniquely determined by pageid and HID identifier.

4.1.2 Node Identifiers

Node Identifiers (NID) are Purple Numbers very similar to the “statement identifiers” of the NLS system. The idea of an NID (Node Identifier) is given in the section 2.2.2. Node identifiers are permanent numbers also called stateful identifiers. Each time the wiki page is edited and saved, the content of the wiki page is scanned for new nodes. Each new node gets assigned a new unique NID which will identify that particular node only. An interesting point here is when a node is edited, e.g., a line is added in middle of a paragraph which already has NID assigned, it should not be considered to be a new node even though the content in that node has changed. If a user wishes to assign a new NID to a node whose content is changed, the user must delete the NID assigned to that node. A new NID will be assigned when the page is saved. In particular, when a user deletes a node, the NID for the node should also be deleted.

One of the requirements of an NID is that it is only unique to the page and not to the entire wiki. This requirement differs from the convention used in PurpleWiki. The reason for this requirement is to ensure that the use of NIDs in a large wiki with many pages and many concurrent users will not encounter performance problems. Creating unique NIDs for the whole
wiki degrades the performance of the wiki because every update must compete for the “next NID” data value in the database.

4.1.3 ViewSpec

A common complaint about Purple Numbers is that they can be a distraction or may even be regarded as intrusive. This is especially true of data-intensive wiki pages. Users may complain even though they like the added capabilities that fine-grained addressability gives them. To address this issue PMWX has the ability to switch between showing and hiding Purple Numbers. This functionality is called Viewspec. This feature is available both at the system administration and user levels. Users of the wiki have the choice of whether or not to use Purple Numbers. Thus Purple Numbers are visible only when the administrator and user wants them to be. Another reason we provide this functionality is to avoid any confusion that Purple Numbers can create for new users who might not want to make use of this feature during their initial visits. If Purple Numbers are disabled by the site administrator at the server side they will not be created. This can typically be done by setting a global flag. This will automatically disable advanced features like transclusion. While if the Purple Numbers are disabled at the client side by the user viewing the web document, they will still be created but won’t be visible to the user who is viewing the web document. So features like transclusion are still enabled.

4.1.4 Transclusion

Transclusion is defined as the inclusion of the content of a document into another document by reference. Transclusion is built on top of the Purple Numbering support which helps to refer to parts in a web document easily and consistently. The goal of transclusion is to get the latest referential data. In other words, if the referenced page is changed, then the transcluded information will be updated on the page even when the page was not otherwise changed. To support transclusion, there must be a consistent way of specifying the content which will be transcluded when one refers to them either using an HID or NID. MediaWiki currently supports transclusion up to a certain level. Users can transclude a whole page or a section of the page. An extension was also developed that would allow users to transclude any node that they want provided that markup has been added to the section for this purpose. Purple Numbers provide this additional markup without any effort by the creator of the page. As web documents can now be addressed at a much more granular level it enables referring to finer portions in a document. In addition, advanced users should be able to transclude a node from a particular version of the page if they do not want the transcluded data to change or if they wish to refer to an older version of the page. This feature is similar to a “cut-and-paste” operation except that the transcluded information is not duplicated in the database.

4.2 Architecture

4.2.1 System Architecture

There are multiple ways in which Purple Numbers can be added to MediaWiki. This section describes the architectural details and how PMWX integrates with MediaWiki. After this various design aspects are discussed, especially the design decisions which had to addressed during development. Purple Numbers can be added to the wiki content in two ways, either by a server-side script or a client-side script. Client-side scripting can be used when Purple Numbers can be generated on the fly every time page is rendered. It is also necessary for scripting to
be available and enabled on the client web browser. While HIDs could be added by a client-side script because they are generated every time the page is rendered, the criterion may not always be satisfied. Accordingly, it is more appropriate to use server-side scripts. Furthermore, NIDs are stateful, so they could never be supported with a client-side script. For the sake of uniformity in the architecture and design and also to allow for users that may not have client-side scripting, we decided to use only server-side scripting. The only feature that is appropriate for client-side scripting is Viewspec.

The system architecture for PMWX is shown in Figure 4.1. The left-hand side represents the functionality of MediaWiki that is used by PMWX. The right-hand side of the figure shows the main parts of PMWX. Because NIDs are stateful (persistent), they are stored in a database. This database would normally use the same database server as the one used by MediaWiki, but they can be different if desired. As shown in the diagram, PMWX adds transclusions, HIDs and NIDs in a pipeline whenever a page is viewed. Like MediaWiki, PMWX is written in PHP and
has been tested using the MySQL database server. Support for other databases will be added later.

4.2.2 Hierarchical identifiers

The notion of a hierarchical identifier is one of schemes of achieving fine-grained addressability. Hierarchical identifiers point to a particular location on a page. As we mentioned in the requirements, HIDs are not immutable i.e. they are assigned to the hierarchical location of the node and not the node itself. If a node moves, its hierarchical information changes and thus its assigned HID changes. HIDs are useful when a page is static or is not changed very often. HIDs are added at the end of the node in a special font using a purple color. HIDs map logically to the physical layout of the document, making it easier to understand. For example HID value (4E8) will always point to the eighth sub statement of the fifth sub statement of the fourth statement on the first level of the document. Note, however, that this hierarchy is the one perceived by the user, not the HTML element containment hierarchy. This is discussed in more detail in Section 5.1.

HID Numbering Scheme

PMWX provides two different numbering schemes for HIDs. The scheme to be used can be selected at the time of installation or later by the administrator who installs and maintains PMWX.

The first numbering scheme is the NLS numbering scheme for HIDs. In the NLS numbering scheme, the identifier for the HID starts with a digit followed by a letter in the alphabet, followed by a digit, and so on. In this scheme, the first node of the document will be numbered “(1)”. The first child of this node will be numbered “(1A)” and the first child of this child node will be numbered “(1A1)”. The second numbering scheme is exactly the opposite of the NLS numbering scheme; namely, the roles of letters and digits are reversed. In this scheme, the first node of the document will be numbered “(A)”. The first child of this node will be numbered “(A1)” and the first child of this child node will be numbered “(A1A)”. 

Figure 4.2: HID Numbering schemes

The first numbering scheme is the NLS numbering scheme for HIDs. In the NLS numbering scheme, the identifier for the HID starts with a digit followed by a letter in the alphabet, followed by a digit, and so on. In this scheme, the first node of the document will be numbered “(1)”.

The first child of this node will be numbered “(1A)” and the first child of this child node will be numbered “(1A1)”. The second numbering scheme is exactly the opposite of the NLS numbering scheme; namely, the roles of letters and digits are reversed. In this scheme, the first node of the document will be numbered “(A)”.

The first child of this node will be numbered “(A1)” and the first child of this child node will be numbered “(A1A)”. 

Figure 4.2: HID Numbering schemes
4.2.3 Node Identifiers

NIDs are very useful for fine-grained addressability of dynamic documents because they are assigned to a particular node in the wiki page and they stay with it for its lifetime. NIDs should stay with the node even if the node is moved around in the document. However, they should not be moved with the node when the node is being copy-pasted into another document, as NIDs are unique to that page only and such an operation could create duplicate NIDs on the other page. Currently we do not have a mechanism for detecting NID duplication but it can be integrated into the extension at a later stage.

NIDs looks similar to HIDs, and when both are being shown, the NID comes before the HID, which follows the node content it is identifying. The shade of purple color used by the NID helps the user distinguish NIDs from HIDs. Figure 2.5 shows a simple wiki page that has both HIDs and NIDs.

4.2.4 NID Numbering Scheme

There is only one numbering scheme available for NIDs in PMWX. The numbering schemes for NIDs follow what is effectively a base-62 numbering system whose “digits” are digits, lower-case letters and upper-case letters (in this order). NIDs are incremented sequentially, and only the largest NID used on each page is stored in the database. This allows one to obtain the next NID whenever a new one is required.

4.2.5 Transclusion

Transclusion is defined as the inclusion of the content of a document into another document by reference. Transclusion is built on top of the Purple Numbering support which helps to refer to parts in a web document easily and consistently. A detailed description of this is given in the section 3.

Since transclusion is using the underlying addressing scheme (HID or NID), it has different pros and cons. The HIDs are stateless and using them to transclude nodes can break the transclusion. When using HIDs branch transclusion is possible. Some of the research questions are, what form does the HTML element take after it is transcluded in the new page, how is the element assigned an identifier (HID or NID), does it retain its previous identifier in the original document, can a transcluded node be transcluded further thus creating a chain of transcluded nodes where only one is the real data node and rest others are references. Some of these issues have been resolved below.

A transcluded node is assigned a fresh number in the page it is moving. Thus only the data part of the transcluded node moves. The transcluded node gets the presentation information from the page it is moving into. As seen in Figure 4.3 node id 1A is transcluded from page1, which is a paragraph element on page 1 and becomes a header element on page 2.

Also as shown in Figure 4.4, list element which is identified by number 1A1 when transcluded gets HID number 1A on page 2.
Figure 4.3: Transcluded Node on page 2 has presentation form compatible with the located in the page it is shown on.

Figure 4.4: Transcluded Node on page 2 has presentation form compatible with the located in the page it is shown on.

Figure: paragraph element transcluded from page 1 becomes header element on page 2.

Figure: list element transcluded from page 1 becomes header element on page 2.

Figure 4.4: Transcluded Node on page 2 has presentation form compatible with the located in the page it is shown on.
Chapter 5

Other Software Used

The idea of using fine grained addressability in Knowledge Management Systems has been around for a long time. Researchers all over the world have been designing efficient Knowledge Management Systems, which make the retrieval and referencing of knowledge easy and accessible. PMWX extension is a reference implementation of the above idea for the most known wiki repository existing today, Mediawiki which is the source code for Wikipedia.

Mediawiki is developed in PHP and it supports both MySQL and Postgres databases as its backend. To have Mediawiki up and running a web server, database server and PHP need to be installed. The most widely known web server, Apache can be used for this. Both Apache and MySQL are freely available for download.

PHP needs to be compiled with MySQL/PostgreSQL capabilities.

PMWX is developed in a way that it provides features on top of the base software. To make the integration seamless PMWX is been developed using PHP and MySQL. However future versions will have support for PostgreSQL as well.
Chapter 6

Reference Implementation

The reference implementation developed has HIDs addressing and node transclusion implemented. Node transclusion currently supports transclusion of text, transclusion of sections and multi-level transclusion. NIDs and branch transclusion work is in progress and is not yet part of the implementation.

The development of the PMWX project uses the object-oriented capabilities of PHP. The class diagram is shown in Figure 2.6. In the class diagram for HIDs, each type of element has its own subclass. This was done so that the assignment of HIDs at the top level can treat all types of elements uniformly. The specific behavior for each type of element is encapsulated in its class. Transclusion has been developed as a separate component. It is left up to the system administrator to enable this feature. In the current version there is support to transclude a given node, and the syntax is powerful enough that the user can specify the version number. There is also support for transcluding the header TOC (Table of Contents) elements at the top of the wiki page. Viewspec is implemented in JavaScript to provide a user with the option to switch between showing and hiding HIDs and NIDs. If the users web browser does not have a script capability, or if the user has disabled the script feature, then the user will not be able to change the setting established by the administrator.

6.1 The HID Implementation

HIDs are generated every time the page is requested by the client. HIDs are added after MediaWiki has converted the wiki markup into HTML markup. This was done to avoid the need for parsing wiki markup since MediaWiki already does this. It has the added bonus that the HTML provided to the PMWX HID processor has a structure that is more easily analyzed. HIDs are assigned to any node that affects the hierarchy of the wiki page as perceived by the user. As we mentioned earlier, each node that is to be assigned an HID must be structured properly. It must either be a child of the overall wiki page or be a child of another node. Nodes (in the HTML markup) that are assigned an HID convey user perceived hierarchical information for rather than presentation information. For example, HTML elements with the tags <p>, <li> and <h1> to <h6> have hierarchical significance to the user viewing the page, but elements with tags such as <b> and <i> only affect the font of the text and are not normally perceived as being of hierarchical significance.

HIDs are assigned as intuitively as possible. For example, <h1> hierarchically supersedes the <h2>-<h6>, <p>, <img>, <li>, <td> tags, and similarly <h6> supersedes the <p>, <img>, <li>, <td> tags. Consider this example of a small part of the HTML of a wiki page:

\[
\text{<h1>Heading 1</h1>}
\]
Figure 6.1: System Architecture
Here the \texttt{<p>} tag is the child of \texttt{<h1>} and so is \texttt{<h2>}. The HIDs assigned to the above elements will be (1), (1A) and (1B), respectively. Now if we have the HTML structure shown here:

\begin{verbatim}
<h1>Heading 1</h1>
<p>Paragraph 1</p>
<h2>Heading 2</h2>
<p>Paragraph 2</p>
\end{verbatim}

then the HIDs assigned to the nodes above will be (1), (1A), (1B) and (1B1), respectively. The numbering scheme is implemented using a local ranking array. Each tag gets its HID by locating the nearest parent node. This operation depends on the type of element, so it was implemented by using a polymorphic method of the subclass corresponding to the element type.

### 6.2 Transclusion Implementation

Transcluded nodes are added every time the page is requested by the client. HIDs for a page are generated on the fly when the client requests the page. This is because the structure of the page may change making previously assigned HIDs meaningless. Since transclusion is implemented such that it uses HID for fine-grained addressability of the HTML elements, transcluded nodes have to be added each time the request for the page is made by the client.

If the page requested by the client contains transclusion syntax, the transcluded HTML element is added just before the page is to be sent to the client. There is support for node transclusion and page section transclusion. The syntax for each is the following:

\begin{verbatim}
<t pageid=id nodeid=hid:num> \end{verbatim}
\begin{verbatim}
<t pageid=id nodeid=num> \end{verbatim}

A page containing transclusion syntax is parsed to get the pageid and nodeid details. The wikitext for the page having the pageid is obtained and is converted into HTML markup by using the MediaWiki parser. To this HTML, HIDs are added using the HID parser. Once the HIDs are assigned to all the nodes on the page, the transclusion parser can get the node with the value as nodeid as mentioned in the syntax.

Transclusion nodes also have HIDs assigned to them and also have a symbol T, which indicates it is a transclusion node. When the user hovers over it, the actual address of the node can be seen. In the case of multiple-level transclusion, where a transcluded node is transcluded further there will be T anchors for each reference.

### 6.3 Algorithms and Data Structure

The way HID numbering is assigned is by having some kind of ranks associated with the HTML elements. HIDs are assigned in a single pass through the HTML elements in the document. Thus the HID number for an HTML element is generated from the information available at that instant, knowledge of the structure of the document till that HTML element is known. During the pass, the current state is stored in an array called as local ranking array.
6.3.1 HID Assignment Algorithm

HIDs are allocated in a single pass through the HTML document. As shown in the Figure 6.3 the HID assignment algorithm is a single pass.

During the pass for a given tag check in the LocalRanking array if value is already initialised
   if value already initialized
      find next HID using getNextHID  -- case 1
      HIDNEXT = HID
   else
      if parent tag having HID initialised.  -- case 2
         if parent tag having HID found
            find the next HID using increaseBreath,
            check for all child Tag having this HID set
            if found
               find the next HID using getNextHID
            HIDNEXT = HID
         else if parent tag having hid not found  -- case 3
            check if there is any child tag with HID initialised
            if found
               get the next HID using getNextHID
               HIDNEXT = HID
            else initialise this Tag with value 'A'.
      
Update the Tag with HIDNEXT value in the LocalRanking array
For all Tags, rank(Tag) > rank(Tags)
Initialise the Tags in LocalRanking array with default value.

Figure 6.3: HID Algorithm

HID Algorithm: The algorithm maintains a local ranking array which gets updated in each iteration. HIDs are generated by using getNextHID and increaseBreath functions. The input to them is an HID value and they generate the next HID. The getNextHID will generate the next serial HID. Some of the examples are: A → B, A1 → A2, A1A → A1B.

The increaseBreath will generate whenever child tag is encountered. As the name says, it increases the breath of the HID. Some of the examples are A → A1, A1 → A1A, A1A → A1A1.

A simple example which shows the state of local ranking array during the pass is shown
below. The 0’s are default values for the elements in the array. A simple document having four HTML elements shown on the left side is considered.

![HTML Document Structure Diagram](image)

State of LocalRanking array

Figure 6.4: HIDs Assignment
Chapter 7

Related Work

There have been many attempts to develop tools to support and to popularize Purple Numbers. These tools were an important influence on the design and development of PMWX. In this section we give a brief overview of the various tools that have support for Purple Numbers in online documents such as web pages, blogs and wikis.

7.1 Xlink

XLink allows elements to be inserted into XML documents in order to create and describe links between resources [11]. XLink was created by Jon Bosak and Tim Bray toward the end of the 1990s. XLink is a powerful mechanism for linking XML documents. XLink allows one to specify bidirectional links, embedded links (i.e., transclusion) and links that have more than two targets. When combined with XPointer [12] and XPath [13], XLink can link and transclude parts of documents, with granularity down to the level of individual characters.

7.2 Purple

Purple [14] is a small suite of quickly hacked tools inspired by Doug Engelbart's attempt to bootstrap the addressing features of his Augment system onto HTML pages. Its purpose is simple: produce HTML documents that can be addressed at the paragraph level. It does this by automatically creating name anchors with static and hierarchical addresses at the beginning of each text node, and by displaying these addresses as links at the end of each text node. Purple is relatively easy to use. The flip side is that it was built for static webpages consisting mainly of text, the web page developer has to run the tool on each web page before it is uploaded on the website as well as whenever it is updated.

7.3 Plink

Murray Altheim came up with an implementation similar in function to Purple (which was mainly in perl), but done in Java around the same time (April 2001) [15].

7.4 PurpleSlurple

Matthew Schneider developed the first HID implementation which generates purple numbers on-the-fly for HTML and text documents which he published in 2002. He had actually worked
on flavors of PurpleSlurple for other document formats like word docs and pdfs, but they did not seem to have been officially released [16]

7.5 PurpleWiki

PurpleWiki [17] is a WikiWikiWeb implementation derived from UseModWiki. It was written by Eugene Kim of Blue Oxen Associates and Chris Dent, and was first released in January 2003. It adds several features to Purple, and modularizes the code for easier development. In addition to support for Purple Numbers, it has a parser that supports pluggable output formats, RSS feeds of recent changes, and transclusion of content between pages. The downside of PurpleWiki is that it only supports NIDs and lacks many features that MediaWiki provides.

7.6 Purple Numbering on Blogs

Tim Bray was the first to put purple numbers (purple hash marks which acted as permalinks, to be exact) to a blog, back in May 2004 [18]. One of the co-authors of this paper, Jonathan Cheyer, has written a plugin to add Purple Numbers for Wordpress [19]. It only supports HIDs. NID are not supported because permanent node identifiers are not stored for each paragraph.

7.7 HyperScope

HyperScope has been a project that Doug Engelbart has been driving over the last decade or so, as part of his OHS effort [20]. A more recent effort has been the NSF funded project (with development work done by Eugene Kim, Brad Neuberg, Jonathan Cheyer et al.) which implements a subset of the functionality of the original NLS system mapped onto the modern day web paradigm. In particular, it supports many of the original NLS viewspecs for viewing documents in different ways. It also supports a number of jump commands which allow you to move between different nodes in a document. The power of the viewspecs and jump commands is that they can be embedded directly into a URL. This allows a user to pass along URLs to another user which point to specific locations in a document with a particular view of that document. The HyperScope site describes the tool as a high-performance thought processor that enables you to navigate, view, and link to documents in sophisticated ways. [21]
Chapter 8

Screen Shots

Purple MediaWiki Extension (PMWX) is been actively used for various collaboration tasks. Screen shots are included here which show the HIDs and transclusion.

The Figure 8.1 shows the HIDs assigned to the html elements on the page. Only the text part of the page is captured here in the image. As can be seen the first node is assigned value 1, which is a header element. The immediate next node is paragraph element, which is assigned HID 1A, and is child of header element.

<table>
<thead>
<tr>
<th>Title</th>
<th>(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purple MediaWiki: Fine-Grained Addressability of Wiki Content</td>
<td>(1A)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authors</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kenneth Baclawski, Viral Gupta, Tejas Parikh</td>
<td>(2A)</td>
</tr>
<tr>
<td>College of Computer and Information Science</td>
<td>(2B)</td>
</tr>
<tr>
<td>Northeastern University</td>
<td>(2C)</td>
</tr>
<tr>
<td>Boston, MA 02453, USA.</td>
<td>(2D)</td>
</tr>
<tr>
<td>kaneb, viral, <a href="mailto:tparikh@ccs.neu.edu">tparikh@ccs.neu.edu</a></td>
<td>(2E)</td>
</tr>
<tr>
<td>Peter P. Yim and Jonathan Cheyer</td>
<td>(2F)</td>
</tr>
<tr>
<td>CIM Engineering, Inc. (CIM3)</td>
<td>(2G)</td>
</tr>
<tr>
<td>San Mateo, CA 94402, USA.</td>
<td>(2H)</td>
</tr>
<tr>
<td><a href="mailto:peter.yim@cim3.com">peter.yim@cim3.com</a>, <a href="mailto:jonathan.cheyer@cim3.com">jonathan.cheyer@cim3.com</a></td>
<td>(2I)</td>
</tr>
</tbody>
</table>

| Abstract | (3) |

Figure 8.1: Screen shot of wiki page showing HIDs assigned to html elements

The interesting observation to be made in the Figure 8.2 is, a list element within another list element is treated as a child and accordingly assigned HID. The html elements with HID 1D and HID 1D1 have a parent-child relation between them.

The Figure 8.3 shows node transclusion. The html elements that have T in purple next to
Figure 8.2: Screen shot of wiki page showing HIDs assigned to html elements

the content are transcluded nodes. Also some of the nodes do not exist on the source page and so the text ‘Unable to transclude’ can be seen.

Figure 8.3: Screen shot of wiki page showing transcluded nodes

The Figure 8.4 shows multiple-level node transclusion. As can be seen the nodes have multiple T anchors. Also the transcluded node is the same, but the presentation on the destination page is different.
**TestPage1**

[ hide purple numbers ]

**transcluded text**

\[ T T \quad (1) \]

**transcluded text**

\[ T T \quad (1A) \]

**transcluded text**

\[ T T \quad (1A1) \]

**transcluded text**

\[ T T \quad (1A1A) \]

Figure 8.4: Screen shot of wiki page showing transcluded nodes
Chapter 9

Conclusions

This thesis solves the problem of providing fine-grained addressability to support large-scale collaborative development of formal document bases by extending a popular and powerful collaboration infrastructure. It has looked at various research problems that were raised in the process of solving fine-grained addressability. After having a good understanding of fine-grained addressability the thesis also looked at how fine-grained addressability can be used as an infrastructure to support powerful feature like transclusion in large-scale collaborative environment.

The thesis has answers to some of the interesting questions like, what level of granularity should the elements in the page be addressable, what elements in an HTML page should have automatically generated anchors, how can multiple sections of a given page be referenced by just referring to one section, how should the text from one page be presented on the other page.

Out next step was to provide a proof of concept and to test the viability of the proposed solutions to the research problems, an extension of MediaWiki that supports fine-grained addressability was developed, which we call Purple MediaWiki Extension or PMWX, . We made a case that fine-grained addressability has been a feature of many types of documents for centuries and that many applications today can benefit from it. The Purple MediaWiki (PMWX) software also demonstrates how fine-grained addressability can be used as an infrastructure for supporting transclusion capability.
Chapter 10

Future Work

The complete study of how NIDs can be used for fine-grained addressability in large-scale collaborative document development has been done. A good understanding has been developed of how NIDs can be used an infrastructure to support advanced features like transclusion. But the design and implementation is left to be undertaken. Also the support for XPath expressions adds a significant degree of complexity to transclusion. The plan is to have the transclusion for NIDs which is relatively easy and then subsequently add support for XPath expressions. XPath expressions gives the extra power to transclude any element on the web. Thus one can transclude parts of a document that are not addressed by either HIDs or NIDs alone and one can transclude parts of documents that do not have fine-grained accessibility at all, such as documents that are not in a wiki.

Once the infrastructure for fine-grained accessibility is in place, we plan to begin work on applications that build on this infrastructure. One such application is the notion of a semantic wiki [22]. Our plan is to support annotations using either folksonomic tagging (such as the del.icio.us web site) or formal ontologies written in RDF or OWL. Existing semantic wikis and tagging mechanisms are limited to annotations at the document level. Fine-grained identifiers allow one to annotate at a much more precise level. We have already developed an initial prototype wiki that allows one to tag purple numbers in the PurpleWiki. We plan to develop a WikiMedia version of this prototype and to extend it to more powerful annotation capabilities.
Bibliography


