Enhancing the AKAP Documents with Prioritised Dynamic Links

Gareth Hughes
IAM Group
Department of Electronics and Computer Science
University of Southampton

October 2000
## Contents

CONTENTS .................................................................................................................................2

INTRODUCTION ..........................................................................................................................3

OVERVIEW OF WORK..............................................................................................................3

IMPLEMENTATION ....................................................................................................................5
  FLOW OF EVENTS ..................................................................................................................5
  INTERFACE IMPLEMENTATION ..........................................................................................5
  MUFFIN ...................................................................................................................................7
  LINKBASE DESIGN .............................................................................................................7
  LINK RESOLVERS ...............................................................................................................8
  5 NEAREST NEIGHBOURS .................................................................................................10

ANALYSIS AND CONCLUSION ..............................................................................................10

FUTURE WORK ........................................................................................................................11

ACKNOWLEDGEMENTS ..........................................................................................................11

APPENDIX ................................................................................................................................12
  SOFTWARE LINKS ..............................................................................................................12
  SETTING UP THE SOFTWARE ..........................................................................................12
Introduction

This report describes work done on providing a tool for dynamically adding links to documents where those links are prioritised and ranked according to the document the user is reading. In order to do this a new system has been developed based on the principles of the Distributed Links Service from the IAM group at the University of Southampton. The system adds a number of innovations to the usual implementation of a DLS as a Web proxy, the primary one being that the system is designed to compute the destination and presentation of each link dynamically. The architecture is designed to allow a programmer to add new ways of computing and presenting links to users depending on the particular document domain and application of the system. The resulting link resolving components form the bulk of the work and are tailored to this particular use. The reusability of the architecture and the link resolution component was a key requirement of the design.

This report should be read in conjunction with the Post Office Research Group AKAP Tools Evaluation Report Part 2 written by Matthew Hall and Sue O’Hare. It takes the results of that project as its starting point. The AKAP document set produced using the Knowledge Capture Tools has been used in this project as Web pages and the word network and document network data produced by Chaomei Chen has been incorporated into the system. This raw data is used for the dynamic computation of each link.

The purpose of the project to is to see if it is possible to use the analysis results in a meaningful way to aid a reader in finding useful knowledge within the documents. Do the links produced help the user to see how the various documents are related as indicated by the analysis work.

Overview of Work

The analysis work done by Chaomei Chen produced a number of outputs. The primary output was a series of 3D VRML worlds illustrating concepts such as document to document relationships and keyword to keyword relationships. This analysis was based on a chosen set of 285 keywords for the document set. The goal of this project was to take the analysis work and try to apply the results back to the documents themselves to see if the analysis helped a reader navigate the documents and find useful content within them.

In Figure 1 below an example of one of the document networks can be seen. Each node represents a document and the clustering shows how the documents are related. Each node is a link that can be clicked on and the user will be taken to the document in question. The VRML world is not easy to navigate and requires a powerful machine to display and manipulate at a satisfying speed.
Figure 1. Viewing the relationships between documents as produced by the analysis work of Chaomei Chen.

A system was designed to add dynamically generated links into the documents as the reader opened them from a Web browser. These links are not just point to point underlined links seen in normal Web pages but are enhanced to have multiple destinations and to represent concepts such as document to document distance. This is done by dynamically adding links to the documents on the chosen keywords whenever they appear in the documents. The technology and techniques for doing this have been developed and refined over a number of years by the IAM group. The unique challenge of this project was to create a system that would dynamically generate and refine the links depending on which document the user was reading. In order to re-use the work done for this project the system was designed to allow a developer to plug in different ways of resolving and displaying links. For instance the AKAP document links need to be displayed according to which document the user is reading to represent the location of that document in the document space. The analysis work produced a number of different sets of relationships between the keywords or documents and the architecture makes it easy to quickly change which link resolution system and data is being used.
**Implementation**

The implementation of the system consists of a Web site of the AKAP documents complemented by an introductory page with a contents listing. When the reader opens a document from the web site the document will be dynamically linked to the documents around it by the use of a link service. This link service is implemented as a Web proxy.

**Flow of Events**

![Flow of Events Diagram]

Figure 2. The order of events between Web browser, link serving proxy and Web server when a user requests a document.

**Interface Implementation**

When a user clicks on a link a menu of destination links pop up. The destinations are ranked and colour coded according to how close the destination documents are to the document being viewed. The design and implementation of this technology was done first so that it would be easier to know what the proxy should generate. Each link anchor has 5 destinations as defined by the data supplied. The strength or priority of that link depends on how far away the destination document is from the document being viewed. A close document has a higher priority, this is indicated by a stronger colour of link. The links are ranked in the menu so that high priority links are at the top of
the menu. It would be a simple matter to filter links so that only links above a certain priority value are displayed. This was not used because there were only 5 destinations per link.

The menu is implemented using JavaScript written for Internet Explorer versions 4 and 5. From this popup menu the user can choose their destination. The priority scheme and colouring is designed using Cascading Style Sheets (CSS) and makes use of the SPAN and ID elements. The data for each menu is stored as DIV data at the bottom of each page. When the user clicks on an anchor the JavaScript loads the data for the menu from the appropriate DIV.

The colours chosen for the experimental system range from a deep red, highest priority, through oranges to yellow. Ten colours were required and it was always known that finding the best colours to use would be a difficult and controversial task. Hence the system was designed so that colours are only defined in the Cascading style sheet file and nowhere else. It is a trivial task to alter that file and change all the colours used in the system. It is also possible to change the colour of the menu background and borders to aid in designing a better colour scheme. There are many other ways to add formatting using the style sheet, for instance a variety of fonts which could vary in size could be used to convey meanings such as priority or whether the link destination had already been visited.

Figure 3. A document viewed in a Web browser enhanced with coloured anchors. If a user clicks on an anchor a menu pops up listing 5 destination documents.
**Muffin**

An investigation into existing Web proxy systems was done to find a code base. Muffin (http://muffin.doit.org/) was chosen as it was already known to the author. Muffin is a free, Java based filtering Web proxy consisting of a filter manager and a large number of filters for performing tasks such as removing cookies, deleting adverts from Web pages and rewriting URLs. One of the filters supplied is a rudimentary DLS-style service called Glossary. It places an anchor on any word in the HTML that matches a simple list given to it. This code was used as a starting point from which to implement the system. A large amount of the time spent programming was devoted to extending and fixing the core code of the existing filter. Hence the time saved by not creating a proxy from scratch was balanced by the need to repair the existing code.

**Linkbase Design**

One of the key features of the design is the use of a database of hypertext links stored separately from the documents. This technology has been used for some time by the IAM group since its design for Microcosm. At its most basic the ‘linkbase’ stores the relationship between an anchor word, e.g. ‘bicycle’ and the document that the user would see if they followed that link. The Distributed Link Service uses the information in a linkbase to add HTML links around each occurrence of that word in every document the user sees.

For example. `<A HREF=http://www.cycling.com/>cycling</A>` would be placed around every occurrence of the word ‘cycling’ in any document the user reads.

The design of the system for this project takes this simple procedure and refines it by dynamically deciding how and if to make a link and exactly how to display that link. E.g. to represent the priority of a destination document by calculating a colour to use. In the case of this system a more advanced linkbase design was needed because for any given word there are a number of possible destination documents.

One of the analysis outputs provided by Chaomei Chen was a table listing each keyword and the 5 most relevant documents along with a priority. This was used to create the linkbase for the system. There are 285 keywords so hence there are 1485 individual links.

The design of a linkbase where each anchor has multiple destinations has not been implemented in the group before. But in the years since Microcosm and the DLS was invented the XLink standard has been created. This is a standard for representing hypertext links in XML for use with XML documents. At the time of writing it is close to becoming a finished standard defined by the W3C. With it are complimentary standards for defining the location of a link in a document called XPointer and XPath. XLink has a concept of a complex link which can include multiple anchors and destinations in a single link object. Therefore the linkbase design for the system uses XLink. A number of standard Java classes for parsing XML and XLink were found and a considerable investment in time was spent understanding and incorporating this code in order to load the linkbase. This investment will be realised in future applications of the system as it will now be a simple to design and work with a new linkbase to match a new application of the system.

It should be noted that the information about priorities is not stored in the linkbase. Only keywords and their destinations. The datasets produced by Chaomei Chen for document-document distance and keyword-document distance are loaded by the PorgResolver and used to compute the link priorities. The PorgResolver knows what document the user is viewing and can look up the priority of each destination document.

A Perl program was written to create an XLink linkbase from the source data.
Link Resolvers

A major improvement of the design had also been carried out so that the system was more useful to the author as a PhD research tool. The notion of a link Resolver was created. This is a module which has sole responsibility for managing link creation and for deciding what information to add to a document. The filter code reads the incoming stream, finds words to link on and writes the document back to the output stream. The Resolver has the job of deciding whether to create a link on a found word and what HTML to actually write. This allows different Resolvers to be plugged into the same architecture for different experiments. Different methods of marking up links in the HTML can be tried or completely new ways of computing links can be written.

The main Resolver written for this project is called PorgResolver. This Resolver creates all of the HTML and JavaScript to create and populate the pop-up menus. Another version of this Resolver has been written that places normal "<A HREF=..." style links into the documents rather than use JavaScript menus. This was done more as a proof of concept that the architecture could work.

The Resolver system is implemented using the Reflection API of Java. This allows for code that finds a class by name, discovers a method in that class and invokes it. As a consequence it is possible to change the Resolver being used on the fly whilst Muffin is running. Figure 4 below, shows the user changing the name of the Resolver in use and an example of the output from the more simple link Resolver developed for the project.

All Resolver classes must support four functions, these corresponds to 4 events in the system.

1. System Initialisation.

When loading the system the Resolver needs to load a linkbase and build a tree of words it will place anchors on. The complicated XLink parsing code and loading of the document-document and link-document relationships is done by the PorgResolver. It is hidden from the main system. The main system just needs a list of words to try and find in the documents. The Resolver must supply this list but how it creates this list is hidden. In this case the anchors in the linkbase are used though this need not be the case in other applications.

2. Adding a document header.

When a document is requested the Resolver is asked to add extra HTML into the HEAD of a document. If the Resolver desires it can make use of this. The PorgResolver adds in lines to download the JavaScript and CSS.

3. Adding a document footer.

Similarly the Resolver is given the opportunity to add to the end of the HTML document. In the case of the PorgResolver this is the point when all the DIV data for the menus is added. During the reading of the document the Resolver will be invoked many times to add links. The Resolver maintains data about what links it has found and the menus already created. If that word is seen again the link information will make use of the existing menu created. Hence each occurrence of a particularly word will link to the same menu data. Considerable time is saved during processing. As many as 6000 anchors are placed in the documents used for the project.

4. Resolving a link.

This is the key activity of the Resolver. Each time a word has been identified as requiring an anchor the Resolver is passed the word and asked to provide the HTML to place into the document. This ensures that all the computation specific to the particular use of the system is separate from the main application.
The Muffin program provides an interface to change parameters to individual filters running in the system. In this case the user is changing the name of the Class being used as a Resolver. The alternative Resolver is being chosen. The filter will then reload and the new Resolver is now running. Any pages now viewed will have their links generated by the SimpleResolver.

There are 2 filters running. DocumentInfo inserts the links to the 5 nearest neighbours and Glossary is the name of the main filter.

This is the same document as that shown in Figure 3 but viewed using the SimpleResolver. The rudimentary way of showing the available links is not meant to be used but just to illustrate the concept of Resolvers.

Figure 4. Changing the link Resolver in use with Muffin and the effect it has on the links produced in a document.
5 Nearest Neighbours
A second filter has been added that adds links at the top of the page of each document that link to the 5 nearest documents in the document space according to the document-document priority data. This can be seen in action in Figure 3 above. The 5 links at the top of the document were produced by this filter.

Analysis and Conclusion
The system now allows the user to browse the collection of Post AKAP documents and view them linked together. 285 stemmed keywords in the documents will be made an anchor for a link each with 5 possible destinations. Now that this system is working satisfactorily it is possible to examine the overall usefulness of the documents, their analysis and the use of keyword linking.

Many of the documents are extremely long. The Post AKAP documents have been used for this work as they matched the analysis data provided by Chaomei Chen. In 42 documents there are 284384 words. For a reader to find the important pieces of knowledge from such an enormous amount of material is an extremely difficult process. This assumes the reader has the time to spare reading them. Therefore it follows that trying to define a set of keywords for the document set is a daunting task even for someone who is a subject expert. The keywords used were chosen by Chaomei Chen who did not make the best choice through no fault of his own. Therefore the top 5 documents for a keyword such as 'telephone' may be of limited use. Needless to say the keywords could be chosen again by an expert and Chaomei could run all of his analysis work again. This time consuming process would presumably yield a much more usable result.

The question is how much more useful could they be? This question is covered in more depth by Sue O'Hare in the previous AKAP Tools Evaluation Report Part 2. In addition to the conclusions of that report it would seem that much of the difficulties in making use of the Visualisation Tools and using the link service produced here lie with the original documents themselves. The major problem is their length. If the documents were decomposed into a vast number of smaller documents (e.g. 1 file for each of the 679 pages) then the analysis work could be more effective in pinpointing the occurrence of keywords in documents. Whilst visualisation of a very large number of documents would pose Chen and users far greater problems it would be to the advantage of the link service.

Choosing and working with keywords is a mechanical method. A more advanced way of working with the documents is to try and define and find concepts in the documents. E.g. choosing a bicycle. This would require more manual intervention in finding the right pieces of knowledge in documents. It may be that some form of thesaurus of concepts or ontology is required. The most advanced version of this technique would be to understand the issues in the documents such as arguments, hypotheses and conclusions.

The assumption being made here is that the documents are worth this amount of work in the first place. Perhaps the lessons should be learnt about capturing the knowledge and the style of documents created and work stopped on this particular set. The author's personal opinion is that they are not, especially the transcripts but will leave that to subject experts to answer.

If another, more manageable document set could found then the author feels that running the full set of analysis experiments could be a worthwhile exercise.
Future Work

Many of the documents in the study are long and, being reports, are full of facts and items of knowledge. The key to future work is to treat the documents as individual paragraphs or sets of paragraphs and apply analysis techniques to them at that level. It is felt that trying to provide a small set of keywords for a 50 page report is asking too much and that the analysis needs to be performed on sections of documents. Work will need to be done on deciding how to fragment documents and at what level of detail to work on documents.

The key to the implementation is to replace the current parsing code with code that treats documents as objects in a tree model. The Tidy code does just that by working with documents using the Document Object Model (DOM). This is also the same model used by Microsoft's JavaScript and standards such as XLink and XPath.

Another possible idea is to try and form trails or guided tours through the documents for a given topic. For instance the user could ask the system to provide a guided tour to all the knowledge in the documents on bicycles. This would be a result of the work done in changing the system to use a DOM. A small navigation web page would take the users to specific paragraphs in documents that are felt to contain the most important facts on that issue.

An area to explore is that of link authoring. A tool could be developed to enable a reader of the documents to create links for themselves that are similar to those currently being generated. A more useful authoring tool may be one that allows users to create a trail or guided tour through the information based on their own analysis of the documents.

The use of very large amounts of link data revealed a number of perplexing bugs in the existing filter code which have been very hard to find and cure. The core code for matching documents has serious flaws which have been dealt with but are not fully understood. The long term plan is to replace this code with an entirely new parser. One has been identified as being that built into the Java version of HTML Tidy (http://www.w3.org/People/Raggett/tidy/, http://www3.sympatico.ca/ac.quick/jtidy.html). The system is for parsing HTML and correcting it. It is supported and highly used code maintained by the W3C and supports advanced features such as XML and the Document Object Model. Hence the parser enables a programmer to work at a word level rather than a character level. This will make it much easier to work into applications. The support for DOM has some very interesting consequences for future work. It will make it possible to load and understand a document on a paragraph by paragraph basis.

In the future it is hoped that the Resolver to use can be decided on a per document basis based on some notion of different document domains. For example documents from the ECS Department Web server would be best marked up using a Resolver that understands the people of ECS and their roles. There is no major impediment to having the filter dynamically choose a Resolver to use as the document is requested. The research issue is how to choose the Resolver.

Acknowledgements

The author proposed this system architecture in September 1999 as part of his PhD work and is very grateful to Jim Ballantyne and Sue O'Hare for an excellent reason to actually settle down and build it.

I am grateful to Chaomei Chen for providing his raw analysis data and patiently explaining it to me.
Appendix

Software Links
The following pieces of software have been used. All are free under the terms of the GNU Public licence or similar. All are on the CD.

Apache Web server (www.apache.org) A free and extremely popular Web server available for many platforms.

Java JDK 1.1. The source code was compiled with Java JDK 1.1.8. It has not been tested with Java 1.2.

Muffin Java Web Proxy (muffin.doit.org) A Java based proxy consisting of a number of filters that can alter the content of a Web page or can alter the HTTP transmissions. The system produced consists of 2 filters for Muffin based on the Glossary filter and DocumentInfo filter.

SAX. (www.megginson.com/SAX/Java/index.html) A powerful and widely used Java API for XML. The basis for the XML parser. SAX 1 is in use not version 2.

XLinkFilter. (www.simonstl.com/projects/xlinkfilter/) A Java class for parsing XLink files that builds on SAX.

Perl. (www.perl.org). Powerful language used to create the linkbase file.

XLink Home. (www.w3.org/XML/Linking) Home page for XLink, XPath and XPointer.

Setting Up the Software
A user will require Perl, Java 1.1, Apache (or another Web server) and a good text editor such as Textpad (www.textpad.com). Some manual editing of the data and configuration files in the system is required such as editing the linkbase and some of the other source files to reflect the name of the user's Web server and the full URL path to the documents on that server. It is a simple matter of using the search and replace tool on a text editor. It is perfectly feasible to set up the whole system on a single computer running Windows 95, 98 or NT and that computer does not to be extremely powerful. There is no reason why the system should not compile on a Unix system though this has not been tried.

The instructions assume a certain level of understanding of Web servers, proxies and Java code. Mainly on the configuration side. It would be advisable to have Java and a Web server running correctly before installing this system.

The CD contains 4 directories, one holds the Web site documents (site), one holds the Muffin source code, docs has this report and other paperwork gathered and 'extra' holds the distributions for all the programs used and the report as well as anything else thought to be of interest or use.

In the following instructions the Web sub-directory 'porg' has been used. There is no reason to use this particular name as it was chosen to make sense to the author. Another Web directory can be used instead though care should be taken to understand the system first before trying this.

For the Web server the 'site' sub-directory needs to be an aliased directory. Eg the following line is from the httpd.conf of Apache.

Alias /porg/ "d:/dev/porg/site/

This will give a Web site URL of http://machine/porg/

The home page of the site is http://machine/porg/index.html
To test whether the menu system is working try the page http://machine/porg/menudemo.html. This will use the JavaScript menus.js and linkstyle.css file. If the menus popup in colour its working. The Javascript has been written for and tested on Internet Explorer 4 and 5.5. It will not work on Netscape.

All of the source code including extra classes is in the muffin-0.9.3a\src directory. You will need to add e:\muffin-0.9.3a\src to the CLASSPATH of the machine or whatever directory is appropriate.

Before running you will to edit a number of files to reflect your system details.

Muffin's configuration file is \muffin-0.9.3a\src\default.conf. There are 3 occurrences of 'http://ivor.ecs.soton.ac.uk/porg/' which you will need to change to reflect your own Web site address. Due to the nature of the system these cannot be file:// type addresses.

In the muffin-0.9.3a\src directory is a sub-directory called 'conf'. There are 3 files in here that are the data for the PorgResolver and were supplied by Chaomei Chen.

docid.txt has a list of id numbers against URLs for the documents. You will need to replace 'http://ivor.ecs.soton.ac.uk/porg/' with your own full URL. The Resolver looks up the URL of the document the user has requested from this list in order to get an id and use it to find out the priority. If all the links are black, ie have no priority then it may be because the URLs of your Web site do not exactly match the URLs in this file.

dpost.idx is the descriptive names of the documents and matches the previous file. It is used to put the description on the menus.

dpost.sim is the data for the document-document network consisting of 2 document id's and a priority value. It is this data that is loaded by the Resolver and used to calculate the priority values and colours.

Note that these 3 files are referred to in default.conf.

Linkbase.xml is the linkbase file which is indicated by the line in default.conf 'Glossary.glossaryfile'. This linkbase has full URL paths to all the documents therefore you will need to search and replace 'http://ivor.ecs.soton.ac.uk/porg/' for your own full URL. There is also a reference at the top of the file to its DTD and this URL should be correct as well.

You may also need to put the jdk1.1.8\bin directory on your PATH though the Java installation may have already done this for you.

To run Muffin change to the muffin-0.9.3a\src directory and type 'java Muffin'

Set up your Web browser to use Muffin as your proxy. In IE go to into the Options dialogue and go to the Connections tab. Set up a proxy of your machine name with a port of 51966, this is the default port that Muffin uses though it can be changed. When you now look at http://machine/porg/ the browser should now make the request through Muffin and you should see Muffin processing the documents.

Note that IE can sometimes think it is cleverer than you and that the document you are viewing does not need a refresh. To force a refresh and make IE retrieve the document again you can press Ctrl-F5 or hold down Ctrl and press the Refresh button.

The linkbase is compiled by a rudimentary perl program in the links directory called xlink.pl. It will require some editing to reflect new directories and the chosen URL of the server before it will run correctly. A competent perl programmer should be able to understand it (and improve it).
The author used the Cortona VRML viewer from http://www.parallelgraphics.com which is free and included on the CD. It can make use of OpenGL and Direct3D. The WRL file shown in Figure 1 is in docs\vrml\dpost.wrl and matches the document-document network used by the application.

This report is docs\my_report\report.doc and was written with Microsoft Word 97.

Gareth Hughes
gvh@ecs.soton.ac.uk
25 October, 2000
IAM Group
Department of Electronics and Computer Science
University of Southampton
http://www.ecs.soton.ac.uk/~gvh/