An open toolkit to facilitate knowledge extraction and document analysis

A 2nd Project Undertaken in conjunction with the AKAP Tools project for the Post Office Research Group.

Gareth Hughes
Intelligence, Agents, Multimedia Group
Department of Electronics and Computer Science
University of Southampton

May 2001
Introduction

The Argentina Knowledge cApture Project (AKAP) attempted to record the useful knowledge and experience acquired by a number of Post Office staff following an assignment to Argentina. A large amount of material was gathered through structured interviews and other techniques. This amounted to a vast pile of documents too large for any single person to read and find useful. Much analysis work has already been done to try and find the most useful knowledge in the documents. The first work was to use keywords and analyse their occurrences in the documents. Relationships were inferred as well as mappings of keyword or document relations using 3D visualisation techniques. The results were of limited use because the keywords were poorly chosen and the documents, some being extremely large, did not lend themselves well to this type of analysis. This authors first project work took the results of this analysis and attempted to link the documents together using the relationships discovered.

This project uses the results of another analysis effort carried out by Kirsty Gallacher. The documents were analysed by hand in a laborious attempt to find the best information in the documents. A document was produced, the ‘Learning Summary’ report, that contains a summary of knowledge found, with references as well as a number of Knowledge Maps. A KM is a hierarchical tree diagram of areas and sub-areas of knowledge covered in the document set. One KM, covering technical areas, is backed up by full references to the actual locations of content covering each area in the document set. This analysis is of high quality and makes good use of the documents.

This project sets out to reproduce the ‘Learning Summary’ report as a living, dynamic Web site. Each part of the analysis has been recreated using a variety of techniques that ensure that the knowledge can be reused and displayed as needed. The Knowledge Maps and references have been written as XML and with links so that
readers can actually see the analysis results properly presented. The goal is to present a superior alternative to a paper based report. The tools can be reused and the analysis data easily edited and improved. The core implementation uses a framework the author is developing as part of a PhD which allows the generation and display of links to be decided dynamically and in context.

The major portion of the work has been in producing new documents that physically represent each element in the technical KM. For instance in the KM there is an area called Track and Trace with a sub-area of Operational Security. In the Learning Summary report there are a number of references to pages in the document set that contain the knowledge in this area. This project has used those references to automatically create a new specialist document on that subject area. This document is, according to the Learning Summary author, the best summary of all the knowledge on Operational Security contained in the document set.

An important goal of the project was to learn about and use the many new standards being developed by the W3C in the areas of hypertext links and data representation. The end of this report features an FAQ on all the different standards and tools used for this project.

This report does not follow the exact order of the submitted work plan. All points on the plan were completed except the supplementary item 10 which concerned building authoring and editing tools. This was not considered to be appropriate.

One of the main conclusions is that a lot of information that was hoped to be captured from the individuals was not done so and many lessons need to be learnt about the interviewing techniques. Therefore any analysis done with this particular document set is going to be of limited value.

**The Web Site**

The site contains the following sections which map to the parts of the original report.

![Figure 1. The Contents page of the live version of the Learning Summary Report.](image)
Learning Summary Report

The main body of text from the report, reproduced as HTML.

Annex 2 - Pre-Analysis Knowledge Map

Annex 3 - Post-Analysis Knowledge Map

These knowledge maps are written in XML and reproduce the diagrams produced in the report.

Annex 4 - Contact List

The contact list has been merged with all other acronyms used in the report to form a reusable XML resource. The Knowledge maps dynamically use the data in the contact list to expand acronyms and initials to their full values.

Annex 5 - Technical Knowledge Map

The Links Between the Knowledge Map and Documents

This is the core of the project and is described below.

The Original Documents

The XML versions of the original documents that this analysis is based on.

Creating the XML Documents

The first task was to convert the supplied Word documents into XML with the least amount of effort. Unfortunately the work was hampered as in the previous project by the poor state of the supplied documents. There was no consistency in the writing style and no use of Word styles used in any document. Subsequently the work took much longer than planned. As observed in the previous project huge amounts of time is wasted because authors have not been trained to use Word properly. Documents are created with no useful mark-up whatsoever and their content cannot be extracted or reused without considerable manual intervention. Therefore the original work and cost of creating the documents is lost to the business as any future use of the data will require the data to be re-edited or rewritten in another form.

The document set was imported into Adobe Framemaker 6 and style information added to the text where economic to do so. A package called Webworks Publisher included with Framemaker 6 was used to export each document to XML. The package creates a complex XML document and makes use of a powerful Cascading Style Sheet to achieve a polished final document that can be displayed on any version 5 browser. These XML documents were used as the basis for the experiments.

The Analysis Knowledge Maps

In the original Learning Report the various Knowledge maps were drawn as shown below. In this project these diagrams were used as the basis for creating an XML document representing the same knowledge. The advantage being that the XML version becomes a useful source of data for other programs and can easily be displayed in a variety of ways by using different XSL stylesheets.

Below is one of the original Knowledge Maps from the Learning Summary report.
This XML fragment is from the Post Analysis Knowledge Map.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="analysis.xsl"?>
<KM
    title="Post Analysis Knowledge Map (Annex 3)"
    version="Product 1.1 Version 1d (post knowledge capture and analysis)"
    project="Argentina Knowledge Capture Project"
    author="Kirsty Gallacher">
    <Area name="Pre Award Phase" id="1">
        <Audience>BPCS</Audience>
        <Audience>NE</Audience>
        <Audience>M&A</Audience>
        <Audience>CPG</Audience>
        <Sub-Area name="Tender/Bid Issues" id="1.1">
            <Knowledge id="1.1.1">PD</Knowledge>
            <Knowledge id="1.1.2">MJ</Knowledge>
        </Sub-Area>
    </Area>
</KM>
```

**Figure 3. A fragment of XML from the Technical Analysis KM.**

This can be viewed on a Web browser as raw XML or by IE6 directly as a Web page if the document includes a reference to an XSL stylesheet to use. An explanation of these technologies is in the FAQ at the end of this report. The result of viewing the above XML is displayed below. Note that the XML above only contains Audience members using their initials. In the page below these are expanded to their full names. How this is achieved is explained in the next section.
The Contact List Knowledge Map

The Contact list knowledge map is a database of people and acronyms used throughout the report. It matches the abbreviations used in the Pre and Post Analysis Knowledge Maps to the people or term. This reusable data source can then be either read by humans, via an appropriate XSL stylesheet, or used by other stylesheets as a data source. The XSL language has a powerful ability to use data from a 2nd XML source whilst processing a document. This is used to create the document seen above to match the abbreviations used in the analysis XML documents to the real people’s names.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="contact_list.xsl" type="text/xsl"?>
<Contacts title="Contact List and Other Abbreviations">
  <Person abbreviation="AM">
    <Name>Andy Marshall</Name>
    <Argentina_Role></Argentina_Role>
    <Current_Role></Current_Role>
    <Contact_Details/></Contact_Details>
  </Person>
</Contacts>
```

Figure 5. A fragment of the XML of the Contact List KM.
Designing the Linkbase

One of the core pieces of analysis contained in the original report was Annex 5 The Technical Knowledge Map. This is a Knowledge Map of all the technical areas covered in the document set. This was followed by 9 pages of listings detailing where in the document set the knowledge was contained. Below is an example of such data.

<table>
<thead>
<tr>
<th>OPERATIONS</th>
<th>KEYWORDS</th>
<th>DOCUMENT</th>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW PROCESSING CENTRE</td>
<td>New Mail Centre</td>
<td>Case Study [AKAP-RO’D-case4-technicalareas.doc]</td>
<td>Rob O’Donaghue</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td>Benchmarking</td>
<td>Case study</td>
<td>Pete Douglas</td>
<td>2</td>
</tr>
<tr>
<td>Lean Team Management</td>
<td>MNI</td>
<td>AKAP Knowledge Questionnaire</td>
<td>Fraser Chambers</td>
<td>3</td>
</tr>
<tr>
<td>Automation</td>
<td>Sorting machinery</td>
<td>Case Study [AKAP-RO’D-case4-technicalareas.doc]</td>
<td>Rob O’Donaghue</td>
<td>2, 3, 4</td>
</tr>
</tbody>
</table>

Figure 6. The contact list KM viewed through a Web browser.

Figure 7. A fragment of the data detailing the pages to find particular items of knowledge in the Word document set.
To make this data useful this information needs to be represented as a set of links, or linkbase, from a point in a knowledge map to a paragraph or paragraph range in one of the source documents.

A way of representing this data in an XML linkbase was investigated and then designed. The data in the first two columns is the data in the XML Knowledge Map. The destinations of each link are to the newly created XML versions of the documents. The original links point to pages in the text. These no longer exist in the XML so for each link a paragraph or paragraph set was specified. This analysis was carried out by hand resulting in the data below.

For each anchor an id has been added and for each page or page range a paragraph number or paragraph range has been created. For certain entries there were multiple ranges which would be represented as multiple link destinations. Each area and sub-area has an id as well.

<table>
<thead>
<tr>
<th>OPERATIONS</th>
<th>KEYWORDS</th>
<th>DOCUMENT</th>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW PROCESSING CENTRE</td>
<td></td>
<td>Case Study [AKAP-RO'D-case4-technicalareas.doc]</td>
<td>Rob O'Donaghue</td>
<td>55 All</td>
</tr>
<tr>
<td></td>
<td>New Mail Centre</td>
<td>Knowledge Transfer Report [AKAP-RO'D-KnowledgeReport.doc]</td>
<td>Rob O'Donaghue</td>
<td>55 All</td>
</tr>
<tr>
<td></td>
<td>CTP</td>
<td>Case study</td>
<td>Pete Douglas</td>
<td>57 2</td>
</tr>
<tr>
<td></td>
<td>Benchmarking</td>
<td></td>
<td></td>
<td>246-249</td>
</tr>
<tr>
<td>Lean Team Management</td>
<td></td>
<td>AKAP Knowledge Questionnaire</td>
<td>Fraser Chambers</td>
<td>58 3</td>
</tr>
<tr>
<td>Automation</td>
<td></td>
<td>Case Study [AKAP-RO'D-case4-technicalareas.doc]</td>
<td>Rob O'Donaghue</td>
<td>58 2, 3, 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6-10 [470-561]</td>
</tr>
</tbody>
</table>

Figure 8. A fragment of data detailing locations of knowledge in the document is marked up with the id numbers of the Technical KM and the paragraph numbers of the paragraphs in the XML documents.

From this a linkbase format could be designed to match the data. It was decided to use the XLink specification for the format of the linkbase.

An example fragment of the linkbase is shown below. It consists of separate elements describing a link in three parts. There are a collection of start points, end points which are joined together by actual links. In the linkbase these are referred to as ‘keyword’, ‘paragraph’ and ‘go’.

A ‘keyword’ represents a point on the Knowledge Map using an XPath statement.

Each ‘paragraph’ entry represents a location in a document as defined by the analysis. The actual destination paragraphs to use are specified using an XPath statement. This statement can be given directly to an engine that understands the standard. When this XPath transform is applied to the original document the result is just the specified paragraphs as XML.
A ‘go’ is an actual link. It states that there is a connection between a point on the Knowledge Map and a paragraph set in one of the original documents. When the linkbase is read in it represents each ‘go’ as a link object to be used and manipulated as required. The separation of the different components of a link makes reuse much easier. It also makes it an easier task to add other types of data into a link object. For instance a link object might also contain a security rating. This can easily be represented as a new component of the linkbase and the code updated to match.

This linkbase design follows the XLink standard. The standard does not specify that a link should have a particular set of components or be laid out as below. It merely specifies the details of the language to use. The linkbase still has to be designed and the code written.

Below is a sample of the actual linkbase showing the variety of elements found.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="linkbase.xsl" type="text/xsl"?>
<!DOCTYPE link SYSTEM "km.dtd">

<linkbase title="Porg 2 Linkbase">
  <paragraph href="http://localhost/porg2/xml/RobO'Donoghue-CaseStudy4-TechnicalAreas.xml" id="55" title="para_title"/>
  <paragraph href="http://localhost/porg2/xml/Pete_Douglas_case-study.xml#//wp:Document[1]/wp:Content//*[(number(@wp:id)&gt;=999246 and number(@wp:id)&lt;=999249)]" paragraphs="999246-999249" id="57" title="para_title"/>
  <keyword href="http://localhost/porg2/km/tech_km.xml#//*[@id='2.1']"
  id="2.1" title="New Processing Centre"/>
  <go from="2.1" to="55" title="New mail centre" arcrole="arcrole"/>
  ...
</linkbase>
```

Figure 9. A fragment of the linkbase created to link items in the Technical KM to paragraphs in the XML document set.

The destination href value is a little complicated.

```
href="http://localhost/porg2/xml/Pete_Douglas_case-study.xml#/wp:Document[1]/wp:Content//*[(number(@wp:id)&gt;=999246 and number(@wp:id)&lt;=999249)]"
```
This actually comprises a URL to a document followed by an XPath statement that an XSL engine can use to extract just the required paragraphs from this document. The XPath statement

```xml
//wp:Document[1]/wp:Content//*[(number(@wp:id)&gt;=999246 and number(@wp:id)&lt;=999249)]
```

says to find all elements in the wp:Document/wp:Content branch where the wp:id value of the element is between 999246 and 999249. Each paragraph has been given an id (wp:id) by Webworks Publisher and the wp:Document style of name elements is also chosen by Webworks Publisher. This statement takes a significant of processing and is further slowed by the considerable size of many of the XML documents. Therefore the processing of the whole linkbase takes some number of minutes.

In the linkbase there are 200 ‘paragraph’ entries, 65 ‘keyword’ entries and 201 ‘go’ entries, or actual links. All were created by hand. This linkbase can be viewed as a document in its own right from the report site front page. See below.

![Image of a web browser viewing the linkbase](image)

**Figure 10.** The linkbase as viewed in a Web browser.

### Creating new documents for a specific topic

The major project goal was to build a program that could read the Technical Analysis Knowledge Map and use the matching linkbase to find the specified paragraphs from the original documents. The paragraphs would be compiled into new documents representing the best pieces of knowledge about a given topic. A fragment of the XML of the Technical Knowledge Map is shown below. The rendered version follows.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="km.xsl"?>
<KM
```
Figure 11. A fragment of the Technical KM as XML.

Figure 12. The Technical KM viewed in a Web browser. Each area and sub-area is linked to a new document of the same name containing all the most relevant information in the document set for that topic.

The Knowledge Map above is similar to the other 2 Knowledge Maps but the XSL stylesheet is more powerful and makes each Area and Sub-Area in the map a link to a document of the same name. The program reads this KM and generates a document of the same name for each Area and Sub-Area. For instance a document called STRATEGY.XML is created containing all of the paragraphs referenced in its sub-areas of Strategy Development and Business Planning. The program also creates individual files for each of these Sub-Areas as well, Strategy Development.xml and Business Planning.xml.

The following figures show samples of the final generated documents.
Figure 13. The generated document for Strategy. This includes all of the paragraphs for Strategy as specified in the analysis as well as all of the sub-areas of Strategy.

Figure 14. The generated document for the sub-area Intellectual Property.
As can be seen the system creates a new XML document that includes links to the source documents. These links will take the user directly to the referenced paragraph so the user can read more of the source document.

**System Architecture**

This section describes in detail the architecture of the system used in this project. The system is being developed for the author’s PhD and as such is not totally relevant in its design to the work being reported here.

The code for the KM reader was written to make use of the work done in the previous project and develop a Resolver for the author’s PhD work. A link Resolver is a self contained library that knows how to compute and display links based on some context. It is designed to be used by a link service to decide what links to return to a user and how to generate them. The basic system used for the previous project was a Web proxy system called Muffin. Into this a new context aware link resolution architecture was developed. The work done in this project has been to further strengthen the architecture design and develop new components to plug into the system. All the code for this project fits into that category. The design is such that the Resolver components do not need to be used as part of a proxy. The Muffin application provided a useful starting point and resource solving many basic implementation problems.

Muffin is Java proxy filtering system for Web pages. It consists of modular filters that can alter any part of a Web page or the HTTP request when placed between a browser and a Web server. This system is used as the container for the author’s work on link resolution.

The Muffin system supports code written as filters. A filter is able to read and alter the content of a Web page as well as the Headers. The usual uses of Muffin are to remove parts of Web pages such as advertisements or applets. Filters can be added at will to the system. The author’s PhD work has used one supplied filter as a basic starting point and rewritten it. The Glossary filter is similar to the Distributed Link Service systems developed by IAM. The filter looks for keywords in the content of a document and will add a HTML link into the document. A linkbase specifies the list of keywords to search for and the destinations. For instance the most basic application of such an architecture is to add links to definitions of words by adding a links into a Web page each time that word occurs. Hence the name Glossary.

The basics of the architecture are below.

![Diagram](image)

*Figure 15. The basic architecture of the Muffin system showing how an individual Filter dynamically passes responsibility for link resolution and creation to a Resolver.*
The Resolver architecture takes the design of a linkservice in a new direction. The filter’s job is to scan the text of each Web page as it is requested by a user. The Resolver will give the Filter a list of words to search for when it starts up. How this list is derived is up to the Resolver. For each keyword specified it plans to replace that word with a link by adding some HTML code for making links. The Resolver is a way of modularising this design. When the filter finds a word in the document it passes responsibility for it to the link Resolver. The Resolver decides whether to add a link and what form and destination that link should take. The implementation of that service is totally up to the Resolver and hidden from the Filter. Resolvers conform to a simple API and are implemented separately from the rest of the system. A major feature of the design is that different Resolvers can be invoked dynamically by the Filter. Hence it is possible to use different Resolvers on the fly as required. How to choose what Resolver to use is an area of work still to be done.

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.

All Resolvers 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 Resolver. 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.

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.

3. Adding a document footer.

Similarly the Resolver is given the opportunity to add to the end of the HTML document.

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 with other contextual information and asked to provide the HTML to place into the document.

The architecture is designed to allow links to be computed and displayed depending on context. The context could be system dependent, user dependent or application dependent. The core goal of the PhD is make each Resolver provide a useful contextual service in its own right. The Resolvers can be dynamically loaded and run as required. Each Resolver is self-contained and all implementation is hidden from the rest of the system. It is hoped that this ‘black box’ approach will enable Resolvers to be re-usable. Resolvers should not just be constrained to link services. The implementation is loosely and historically based on a link service but the goal of the work is to ensure that implementation is general enough for them to do more than
that. The PhD plan is to develop a number of diverse applications of the which will
force the design to evolve and improve. This project has provided core programming
components to aid in that goal.

**XMLFragmentResolver**

For this project a Resolver was developed to fully support the final specification of
the XLink specification and to able to easily adapt to new linkbase designs. The
function of the Resolver is to apply XPath statements to documents and return the
resulting XML. In this case the Resolver returns the paragraphs specified by the
Learning Report author.

The KMReader program uses a Resolver to perform the main function of the program.
In this case the Resolver is being used outside of the Muffin system demonstrating
that Resolvers are standalone libraries. The KMReader is given a KM and a linkbase.
It initialises the Resolver with the linkbase and proceeds to read the KM. For each
area or sub-area in the KM it invokes ResolveLink with the area id. This causes the
Resolver to find all destination paragraphs for that keyword id. For each link found
the Resolver applies the XPath statement to the document by invoking the Xerces
engine. The Reader creates a new XML file with the name of the area and adds each
XPath transform result. The result of this process is a set of new documents.

Because of the volume of material involved and the slow XPath engine the process
takes a number of minutes to run. It is not currently feasible to use the system
dynamically with any useful speed. However, because the code is written into a
Resolver it does work perfectly (albeit slowly) as a link Resolver in the Muffin proxy
developed in the first project.

**Transclusion Scenario**

The following scenario gives a use for this technology, the current version does not
exactly match the example but the example makes it clearer.

A simple XML document has been authored following an analysis the document set.
In the document they can refer to paragraphs in the linkbase using the paragraph id.
When this document is passed through the proxy the linkservice will call the Resolver
with that paragraph id and the Resolver will replace the reference with the paragraphs.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="../document.xsl" type="text/xsl"?>
<wp:Document
    xmlns:wp='http://webworks.com/publisher/xml/schema'
    title="Executive Issues concerned with Strategy Development">
    <wp:Content>
        <Heading2 wp:class="Heading2" wp:id="999263" wp:style="display:
            block;" wp:type="para">
            This document contains an executive overview of the most
            important lessons learnt on the issue of strategy development
            during the Argentine experience.
            Richard Barnes puts it most succinctly.
        </Heading2>
        <paragraph id="4" />
    </wp:Content>
</wp:Document>
```
Figure 16. An example of an XML document that an analysis author could create. It includes references to paragraphs using the linkbase notation. A processing engine would replace the reference with the actual paragraph.

When this document is opened through the Muffin proxy using the XMLFragmentResolver `<paragraph id=4/>` will be replaced with the relevant paragraph pertaining to this subject as specified in the linkbase. A new document is easily created using simple references to the original document.

If this report was an XML document and viewed using the technology it would include links to the extracts from the previous report and other included documents. The link service would include the paragraphs when delivering the document to the user. The Resolver architecture could then be used to tailor this content to the user providing contextual tailoring of the content based on any number of factors. If the factor can be represented in a program then it should hopefully be able to be written as a Resolver.

Other Resolvers and Contexts

The XMLFragmentResolver has components that can be reused for many other Resolvers and applications. The linkbase it supports is implemented to conform to the XLink standard and the code is a self-contained component. The linkbase design and matching code is extensible.
One example under development is a design where each link has a start anchor, a
destination and a time span. A timespan consists of 2 dates forming a range. The
objective is to develop applications in which a linkservice supplies appropriate links
in context of the date of the document the user is reading. The date range components
of the linkbase design have been added to the basic design and extra functionality
written to allow a link service to query for links by date.

There are at least 2 applications of this with online archives. The simplest one is to
place links in documents which match the age of the documents. Therefore if a link is
valid for documents between 1998 and 1999 then it will only be returned if the
documents age is within that range. A more sophisticated example would be to allow
the user to view an archive and be able to ‘turn back the clock’ through some
interface. The archive and link service would show the state of the system at that time
by only showing documents created up to that point and only showing links to match.

The concept of a Resolver that understands time spans can be expanded to more
ambitious applications. The linkbase itself forms a timeline of information. It is
intended to expand on this area with some novel applications of links with a time
context. Little other research is done in this area as the majority of work uses time
based media such as video or audio. This work is concerned with time information
contained in the metadata of the document.

Another Resolver is one which understands the users status within an organisation.
This could take account of user profile information such as status, interests, age or
project. Within the Department of Electronics and Computer Science a Resolver can
make use of the internal personnel database containing information such as research
group and whether the person is an undergraduate, researcher or academic. A
Resolver is under construction which uses an LDAP service to obtain this data. A user
will be able to log on to the linkservice and from then on all links will be tailored to
them.

This Resolver is application and domain specific. It will only be useful within the
department and only truly relevant when applied to documents served from the
department. Therefore the context component is highly specific and the design does
not need to be generalised to encompass all possible uses. If another workplace needs
a matching Resolver then it should be slightly recoded to match. This includes code
and linkbase design.

The ability to dynamically chose which Resolver to use on a per document and per
link basis opens up opportunities to experiment. Other areas of context research in the
IAM research group have focussed on how to decide what is the current context of a
document. The work used content analysis techniques to decide what the subject of a
document was and then served links accordingly. One linkservice and link design was
used for the system. The work presented here makes no attempt to automatically
determine context but provides a mechanism to provide new linkservice link designs
to determine and deliver links.
Conclusions

The interviews of Post Office staff connected to the project with Argentina produced an overwhelming amount of information with large quantities of noise. Finding the useful information and making use of it has been tackled by a total of 4 projects. They have shown that automated document analysis techniques can work but need good quality and preferably short documents to work well. A person has done a better job of analysing the documents to provide a good summary of the knowledge stored. This project has brought that analysis to life and will allow the results to be better utilised by the organisation. The subject-specific virtual documents created by the system can be distributed accurately to the people who would benefit from them the most.

A number of techniques have been developed for displaying the results of document analysis using an open linking architecture and state of the art document standards. A system has been developed to link analysis text such as knowledge maps to the sources of that analysis - the paragraphs in the original documents. This has enabled a program to be written that can use the links in a knowledge map to extract paragraphs from source documents and amalgamate them into new documents. From this a set of new documents have been created which represent the most important captured knowledge in the document set. The new documents are small, subject specific executive summaries of the knowledge stored in the original documents. This facilitates analysis and also makes it feasible for the lessons learnt to be distributed to a wider audience. The lessons learnt in a specific topic can be quickly fed back to the people responsible for those areas.

The project has been hampered by the poor quality of the supplied documents. This should be a concern to the Post Office as documents written badly are a great cost to an organisation. Unless the documents include mark-up or styling information the data is ‘lost’ and cannot be reused without more work. Retyping and reworking of data is a great cost to a business. This project has shown that if documents are created well in the first place it is possible to make much greater use of the knowledge they contain.

A major effort has gone into understanding and using the new and emerging standards of XLink, XPath, XSLT and XPointer as an attempt to understand them and produce reusable code. Many of the standards are so new that they are not highly supported at the time of writing and working with them has been difficult. The XPath engine slows down the current implementation. It is being hampered by the size of the documents it has to deal with. A much faster solution would be to ignore the XPath standard and make use of the Perl language to perform the text extraction process using regular expressions. It should be remembered that the goal was to understand and embrace emerging standards so the speed of the final system is not an issue for this project.
FAQ
This set of Frequently Asked Questions explains the many new technology standards and some of the major code libraries learnt and used in this project.

1. What is an XML parser?
   An XML parser is a programming library for working with XML. There are two styles of implementation and both have their uses. The first is DOM (Document Object Model), this loads the XML and presents the programmer with a tree-like structure they can work with using appropriate functions. This is useful where the programmer wants to do things such as find the 3rd occurrence of an element. The 2nd model is SAX (Simple API for XML) which uses an event-driven model. The programmer sets up a parser and listens for events such as the start of the document, the start of an element etc. It allows a developer to write code to only respond to certain points in an XML document. Certain parsers supporting SAX do not need to load the whole XML document into memory to work which has advantages for dealing with very large XML sources.

2. What is XPath?
   XPath is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer. It is a standard of the W3C.

3. What is XSL?
   The Extensible Stylesheet Language is a near complete standard from the W3C. The introduction to the standard describes it thus...
   XSL is a language for expressing stylesheets. It consists of two parts: a language for transforming XML documents, and an XML vocabulary for specifying formatting semantics. An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary.
   An XML parser such as Saxon or Xalan or MSXML will take an XML file, an XSL file and apply the stylesheet to the XML to produce new XML or any other output required such as HTML.
   Microsoft Internet Explorer 5 has an XSL capable parser built in called MSXML. The version of the MSXML parser that is shipped with IE 5.5 is version 2. In order to view the documents created with this project MSXML 3 is required and is something that must be downloaded and installed. Internet Explorer 6 will ship with MSXML 4.

4. What is XSLT?
   XSLT is a language for transforming XML documents into other XML documents. It is a standard of the W3C. XSLT is designed for use as part of XSL, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.
   XSLT is also designed to be used independently of XSL. However, XSLT is not intended as a completely general-purpose XML transformation language. Rather it
is designed primarily for the kinds of transformations that are needed when XSLT is used as part of XSL.

5. What is XLink?

XLink is a specification from the W3C for describing linkbases. It allows elements to be inserted into XML documents in order to create and describe links between resources. It uses XML syntax to create structures that can describe links similar to the simple unidirectional hyperlinks of today's HTML, as well as more sophisticated links. It has its roots in the Hypertext community and cites Microcosm as one of its influences. It has just become a finished standard.

6. What is XPointer?

XPointer, which is based on the XML Path Language (XPath), supports addressing into the internal structures of XML documents. It allows for examination of a hierarchical document structure and choice of its internal parts based on various properties, such as element types, attribute values, character content, and relative position. It is a nearly complete W3C standard.

In this project the XPointer standard could have been used to specify the links. Support is currently too sparse to do this as there are no libraries that support the XPointer syntax. Instead the URLs used in the linkbase are XPointer in style but actually XPaths.

7. What is Xalan?

Xalan is a Java XSL processor for transforming XML documents into HTML, text, or other XML document types. It implements the W3C Recommendations for XSL Transformations (XSLT) and the XML Path Language (XPath). It is produced by the Apache project.

Xalan is the engine used by the Resolver to transform an original document into just the required paragraphs.