Large scale acquisition and maintenance from the web without source access

Thomas Leonard
tal00r@ecs.soton.ac.uk

Hugh Glaser
hg@ecs.soton.ac.uk

University of Southampton
Southampton SO17 1BJ UK

ABSTRACT

Although different web sites structure their pages differently, the pages within a single site are often generated from a database and have a regular layout from which it is possible to extract information automatically.

Dome is a visual tool for manipulating tree-structured documents. It can import and export in XML or HTML formats, making it ideal for harvesting information from web pages. Editing is performed using a direct manipulation interface and the operations are recorded for later playback.

The knowledge extracted from a web page may be updated by replaying the recorded sequence when the source page changes. The same sequence can be applied to other pages with a similar format, and facilities are provided to batch process a large collection of pages in one operation.

In this paper we describe how Dome may be used to extract knowledge from web sites in such a way that the extraction process may be reliably replayed.

Keywords

Knowledge acquisition tools, Programming by demonstration systems, Programming by example, Visual languages, XML editors

INTRODUCTION

Recent interest in the semantic web, Tim Berners-Lee and others' vision to make web pages' inherent knowledge directly accessible to machines, has produced a desire for knowledge extraction systems which work on existing web pages.

While, in the longer term, Natural Language Processing (NLP) tools of great complexity are needed, these tools do not yet exist. In the medium term, or when a high level of confidence in the accuracy of the results is required, a more 'programmed' approach can be used.

There are a number of tools available for specifying the automatic extraction of knowledge from web pages, but they usually require the user to enter complex query commands.

For example, Web-OEM allows HyperText Markup Language[7] (HTML) documents to be queried like a relational database, using Structured Query Language (SQL) syntax. It also provides a mechanism to create Extensible Markup Language[6] (XML) files from the results by specifying a template, as in this example (taken from [2]):

```
CONSTRUCT<EMAIL>x1.text</EMAIL><TEL>x2.text</TEL>
FROM Page:p, Table:t, Text:x1, Text:x2
WHERE p.title="My home page"    AND
        t IN p.structures.*     AND
        x1=t.row[0].elements[1] AND
        x2=t.row[2].elements[1]
```

In this paper, we describe a visual tool which can perform such tasks easily using direct manipulation, while still allowing the operation to be replayed later.

In particular, we will show how knowledge may be extracted from an entire site, and how that knowledge can be kept up-to-date.

DOME

Dome is a visual language which focuses on manipulation of tree-structured data. This makes it ideal for processing XML and HTML documents.

The program may be used simply as an editor, and supports the familiar editor operations such as cut, copy, and paste. Once the editing of documents using these direct manipulation operations is mastered, the user may easily string operations together to form programs.

The main window is divided into three parts (see figure 1):

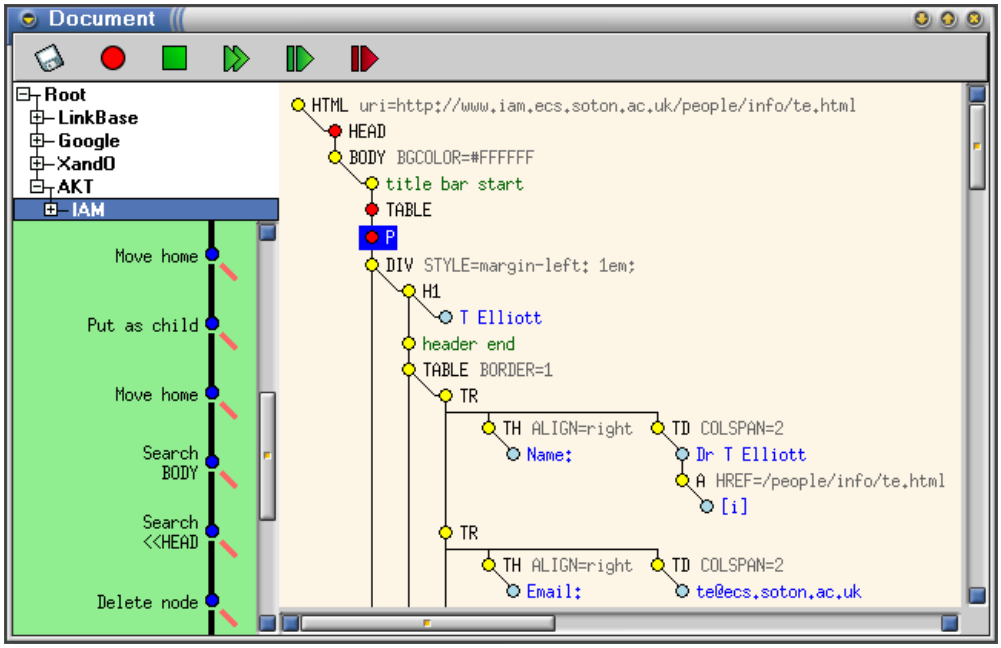

Figure 1: Dome's main window, showing a page from our department's site. The darker nodes in the document (`HEAD`, `TABLE` and `P`) are *collapsed* nodes — this feature allows unimportant elements to be hidden, reducing screen clutter.

The Document The main area, on the right, shows the data that the user is editing. In our case, this is the HTML of the web page, showing its tree structure.

The layout should be readable to anyone who knows HTML. A vertical line represents a sibling relationship between nodes, while a diagonal line indicates a parent–child relationship.

The single exception to this rule is the 'TR' element, which is used to create a row of cells in HTML. Dome lays out the child nodes of a TR element horizontally to save space and to make it look closer to the way it appears in a browser.

The Programs List Each sequence of operations that the user has recorded is displayed in the top-left corner of the screen. The programs can be organised into a hierarchy if there are a large number of them.

The tree of collapsable nodes behaves like the directory list in Microsoft's Explorer program.

The Program Display The operations of the currently selected program are displayed below the program list.

This is a control flow diagram — control normally passes downwards along the dark lines. The fainter diagonal lines are used when execution of a operation fails for some reason. A dotted line (as seen in figure 2) indicates a breakpoint, where execution stops to allow the user to examine the state of the system.

The user can also use this area to correct mistakes in recordings and to record alternative cases.

The most important operation for our purposes is that of selecting a piece of information. There are three common ways of selecting a node in the document:

1. A structured relative move is performed for any node clicked on. Dome records the operation as an XPath[5] which will select that node relative to the current node. For example, "Move to the first cell of the next row.".
2. A non-structured text search — for example "Find the word 'Name:' anywhere in the page.".
3. A structured search which also requires a literal match of the text of the node clicked — "Move to the first cell of the next row, which must contain the text 'Name:'.". This is done using a vendor extension of the XPath syntax.

Although all three methods may be used to select the same node, choosing the correct method is crucial to making the operation replayable.

The first is the easiest and is quite tolerant of changes to document structure. It is sufficient for many purposes, especially if the document's structure is unlikely to change.

Either of the other two may be performed first to make the search more reliable or more strict. Consider a table row containing two cells: the literal string "Name:" and the name itself. By using method 3 to select the literal string and then using method 1 to select the name itself, the recorded sequence will not be fooled by a table with a new first row – it will fail with an error instead of selecting the wrong node.

By contrast, using the second method to search for the string "Name:" and then using method 1 to select the actual name will still work correctly even if a new row is added. However,

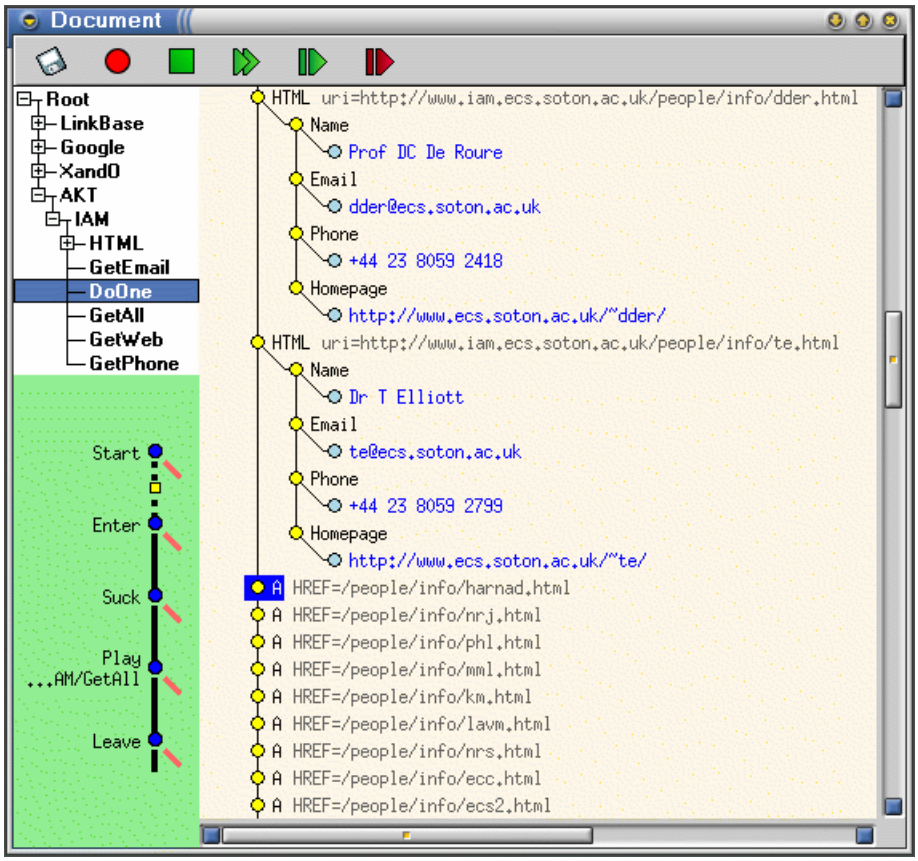

Figure 2: Dome in the middle of processing a research group's site. A page containing links to all members of the group has been automatically fetched and the links extracted from it. Each link in turn is replaced by the knowledge extracted from the linked page. The program displayed is the 'enter–fetch–process–leave' program — it was stopped for the screenshot by setting a breakpoint (the dotted line) while the program was running.

it is also more susceptible to selecting the wrong node altogether if "Name:" appears somewhere else in the document.

PROCESSING ONE PAGE

In a typical editing session, the user will load a web page from the site of interest into Dome. Then, for each piece of information that needs to be extracted, they will record a program to extract that information.

For example, if the aim is to collect product details, the user may create a program called 'Name' by performing whatever actions are required to extract the product's name. This is often as simple as scrolling down to find the name, selecting it, and then using copy and paste to bring it to the top of the document, perhaps placing it under a new element node called 'Name'.

The process will then be repeated to create programs called 'Price', 'Order code' and so on. Once all the data have been collected, the rest of the document is deleted, leaving a neat XML record to be saved out.

Although it is possible to record all the actions in a single program, we find that it is easier to cope with errors (such as a product with no order code) if each piece of information is extracted separately.

To extract information from a similar page, the user may load the page in and click on each program in turn to run it. Once confident with the function of each program, the user will normally start recording a new program and then click on each of the previous programs in turn to create a master program that processes a whole page in one go.

PROCESSING A WHOLE SITE

When processing a whole site, two extra features of Dome are useful:

- Dome includes facilities to fetch a page referenced by a Universal Resource Identifier (URI) in a document. It does this by replacing the anchor element node (`A`, for example) with the contents of the page fetched.
- Dome allows a subnode in the document to be treated, temporarily, as the document root (called 'entering' the node). 'Leaving' the node returns to the previous root node.

To process an entire site, the following steps are typically used:

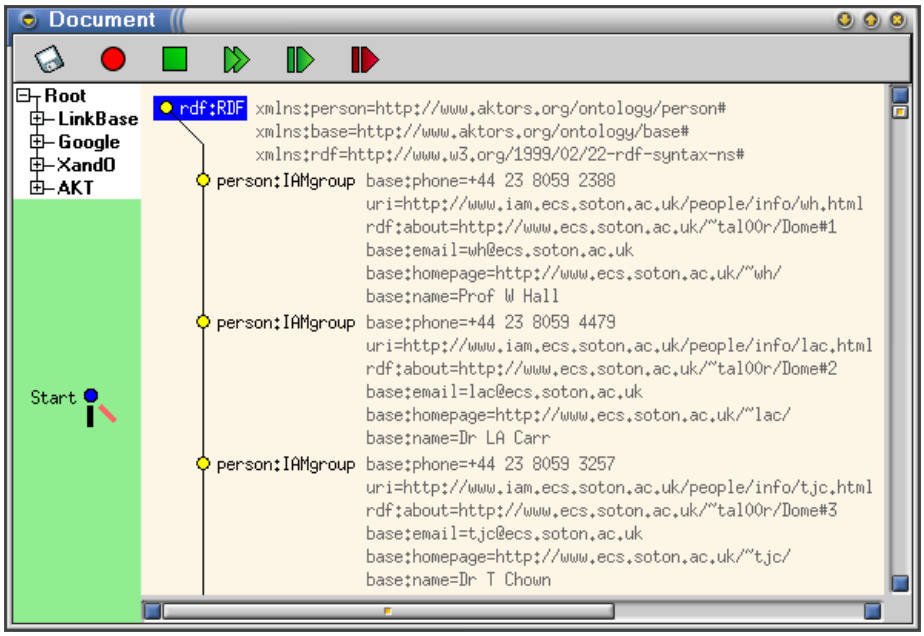

Figure 3: The extracted knowledge, converted to RDF.

1. Load a page which references all the subpages to be harvested.
2. Record a program which extracts all the relevant 'anchor' nodes from that document.
3. Select the first node and record a program which enters the node, fetches the HTML document, runs the program which processes one page, and then leaves the node. This has the effect of replacing the reference to the page with the information extracted from the page.
4. Select the remaining nodes and 'map' the previous program (Dome will run the enter–fetch–process–leave program on each of the selected nodes).

This generates an XML document which is a list of pages and their extracted information.

When each subpage is fetched, Dome records the URI it used by adding a 'uri' attribute to the new element. This is done mainly to allow relative URIs within the fetched document to be resolved, but for our purposes it means that each record in the XML file can be used just like the original anchor — that is, we can rerun the 'map' operation, without any modifications, to update every record.

This is useful if extracting the anchor nodes had to be done manually. If processing the index document is trivial then it is, of course, better to run the whole thing again from the start to cope with newly added or removed pages.

ROBUSTNESS

It may be that, while processing a site, Dome hits a page which has a structure different from that expected. For example, a product which has no order code (perhaps because it is out of stock).

In making the system more robust to changes in the structure of the document there are two points to consider:

- Making sure that any significant change is detected and reported to the user. The system should not simply generate incorrect output. This is best achieved using a structured-literal search, as discussed previously.
- Handling structural changes when they are detected.

  In this case, the program will fail and execution stops at the point of failure. Dome displays the steps of the program that failed and asks the user if they would like to record a 'failure case'. The user agrees and proceeds to take the required actions (perhaps by selecting the 'out-of-stock' text, instead of the missing order code element, and bringing that to the top).

In this way, the user builds up a list of exceptions which allow Dome to process the entire site.

EXPORTING THE RESULTS

Dome can be used to export the results in a variety of formats. If some format other than plain XML is required, another program may be used to convert to that format (still using Dome). More usefully, several programs may be employed to export the same knowledge in a variety of formats.

For example, it is very easy to convert a list of XML records into an HTML table. Add the required HTML elements (`HTML`, `HEAD`, `BODY`, etc) and then use Dome's save-as-HTML feature to create a document ready to publish on the web.

For use in knowledge systems, records may be converted to Resource Description Framework[4] (RDF) format, as shown in figure 3, perhaps using a semantic vocabulary such as Dublin Core[1].

CURRENT STATUS

Dome is a research prototype, currently implemented in the Python[3] programming language, on Linux. It uses the GTK[8] toolkit for the user interface, and the 4Suite XML tools[9].

It has already been successfully used to extract information about researchers from a number of UK sites. The examples in this document are taken from the web site of one of our department's groups. As a rough speed guide, extracting personal details from the 122 individual web-pages linked from the group's 'Complete List of People' page takes around 20 minutes on a typical desktop system.

Much of this time is spent in network communication and in importing the HTML, which is done in two stages. The HTML is first piped through the Web Consortium's 'Tidy' program to correct broken HTML, then the result is parsed using the 4Suite tools.

CONCLUSIONS AND FUTURE WORK

In this paper we have shown how Dome may be used to extract information from web pages into appropriately formatted XML documents. We have seen how to process many pages automatically and we have looked at ways of making the extraction process robust to changes in document structure.

There are several other areas where parsing structured web pages is useful. Metasearchers search the web by querying many other search engines and combining the results, but since they may have to perform millions of searches a day, speed requirements dictate the use of hand-coded parsers. However, Dome is well-suited to tasks such as creating a news roundup by taking headlines from a number of other sites, as this only needs to be done every few minutes.

Some aspects of Dome may be improved — for example, there is potential for a considerable speed increase if web pages could be retrieved in parallel with processing operations.

Object-oriented features may be added at some point, so that 'programs' become 'methods' that work on a class hierarchy of element tags. While this is not immediately useful for HTML, it will improve Dome's ability to handle the structured XML records produced from the HTML.

Even in its current state, we feel that Dome is already a useful tool for anyone wishing to process web pages in a structured and repeatable way.

1. REFERENCES

[1] The Dublin Core Metadata Initiative Available at `http://dublincore.org/`.

[2] Iocchi, Luca. The Web-OEM approach to Web information extraction. *Journal of Network and Computer Applications* (1999) 22, 259–269.

[3] The Python programming language. `http://www.python.org/`.

[4] The World Wide Web Consortium. Resource Description Framework. Available at `http://www.w3.org/RDF/`.

[5] The World Wide Web Consortium. XML Path Language (XPath). Available at `http://www.w3c.org/TR/xpath`.

[6] The World Wide Web Consortium. Extensible Markup Language (XML). Available at `http://www.w3.org/XML/`.

[7] The World Wide Web Consortium. HyperText Markup Language. Available at `http://www.w3.org/MarkUp/`.

[8] The GIMP Toolkit. `http://www.gtk.org/`.

[9] Fourthought, Inc. Open source XML processing tools. Available at `http://4Suite.org/`.