Towards Improving Automation in the World Wide Web

Simon Dobson and Victoria Burrill

Computing and Information Systems
Rutherford Appleton Laboratory
Chilton, Didcot, Oxfordshire OX11 0QX.

e-mail: {sd, vab}@inf.rl.ac.uk
WWW: http://www.cis.rl.ac.uk/

Abstract

Much current work on World Wide Web (WWW) concentrates on improving the presentation of documents. There is also a complementary need to search for information according to complex criteria, and to extract information automatically from pages. We describe *lightweight databases*, an experiment in adding semantic mark-up to web pages, and show how the technique can be used to provide limited-performance database functionality without undue overheads. We describe sample applications including cross-server databases, client-side searching and generating printed documents from hyperlinked pages. We conclude with some potential directions for future work on automation.

1. Introduction

The trickle of interest in wide-area distributed information systems has grown to a torrent with the emergence of the World Wide Web (WWW) as a *de facto* standard for sharing information across Internet. There are now several thousand servers throughout the world run by academic, governmental and commercial organisations, providing access to several million pages of information.

The prime driving force for current WWW development is the need to make the system more accessible to business and private users, empowering them to make use of the information available through WWW and stimulating them to add to it. In particular many businesses feel the need to make material available electronically with the same “glossy” quality as their paper brochures. However another important strand of research is beginning to emerge forcefully: the need to improve the machine-readbility of WWW, making the information in it more amenable to automatic searching and processing.

Our aim in this paper is to describe briefly some experimental directions we have been pursuing in these areas, and to provide some pointers to possible future developments in WWW automation.

2. Emerging Properties and Problems of WWW

WWW is supported by a few key principles which, although simple in themselves, combine to produce an extremely rich information space. Perhaps the most important characteristics of the system are:

- *use of hypertext*, allowing information to be connected using links which users may traverse to navigate through the information space using a simple point-and-click metaphor;
- *a standardised mark-up language* the design of which is now a major bone of contention within the community but which nevertheless allows text, graphics and hyperlinks to be combined quickly and simply;
- *decentralised publication* whereby any organisation or individual with an Internet connection can make information available to the world; and
- *subsumption of other services* so that, within a single framework, one can access hypertext, FTP-able files, news, mail, and all other Internet services through the mechanism of Uniform Resource Locators (URLs)[2].

It is important to realise how these factors contribute to the web as a whole. The use of hypertext is perhaps the key feature which caused WWW to succeed as a global information architecture where previous efforts, such as gopher and WAIS, failed to capture the world's imagination. The adoption of the hypertext mark-up language (HTML)[3] provided the expectation of (reasonable) stability which encouraged organisations to establish servers and contribute documents. The ability to run a server without any central administrative overhead has resulted in a large (and ever-increasing) server population. Finally the web subsumes all other services within a common framework, allowing it both to encompass what already exists and to expand to take advantage of new technologies.

However the unrestrained growth of WWW is beginning to show signs of strain caused by limitations in current practices. Indeed, it is not going too far to suggest that WWW may fall victim to its own success!

The web is currently growing in three directions: the population of servers, the number of pages at these servers, and the amount of inter-linking of pages. The amount of available information and the extent of cross-referencing are growing exponentially. This in turn makes it difficult for anyone to maintain an up-to-date picture of their own web, let alone any other. An immediate consequence is the difficulty in acquiring information from WWW which is *complete* and *consistent*.

The problems revolve, in our view, around three centres:

- the use of a simple client/server architecture;

- the need for human intervention in most page maintenance operations; and
- the lack of semantic information about page contents.

The use of client/server means that servers are wholly independent and do not co-operate in the maintenance of a consistent information space. A human moderator must update pages and links, a task which becomes increasingly more difficult as the web grows. Since pages do not provide machine-readable descriptions of their contents, it is difficult to use automatic searching and summary techniques to process pages to generate alternate views.

We believe the answer to many these problems lies in increasing the level of automation in WWW by extending current technologies and practices so that they better cope with scaling effects. For this paper we shall concentrate on an experiment which we have conducted into improving search techniques for WWW pages.

3. Lightweight Databases and Semantic Mark-up

Since providing information to the world is the *prima facie* reason for WWW's existence, problems which impede users' full access to the available information are serious indeed. Any proposed solutions must be sensitive to the need to scale-up along with the web in order to accomodate future growth in the information space.

Database Gateways

A number of authors have investigated using WWW to access a relational database management system (RDBMS) using a “gateway” to generate queries and convert the results into HTML. This is a good example of the web's subsuming another information service – in this case an RDBMS – seamlessly into a single framework.

A typical case is where an institution extracts “contact” pages for its employees directly from its personnel database, possibly according to some selection criteria. A user retrieves a URL – probably by selecting a hyperlink from another page – and the WWW server transparently generates a query to the RDBMS. This extracts the person's details and re-formats them into HTML for transmission back to the user. The page is *virtual* in the sense that it has been created in response to the request rather than simply being retrieved (figure 1), although from the user's perspective it appears that a page has been retrieved as usual.

Most gateways are thus transparent to the user[7]. This is both a strength and a weakness: it allows users to access seamlessly other information sources through WWW, but may provide only limited access to the underlying RDBMS by permitting only limited number of pre-defined (and invisible) queries. Moreover it enforces a division of work whereby the server performs the query, which may cause the server to become excessively loaded.

Lightweight Databases

An alternative approach is to take the view that the *pages themselves* are the database. One impresses a database structure onto pages, providing an alternative view on the information they contain and allowing them to be accessed using the usual database formalisms. This is the experimental *lightweight database*[6] approach.

One may observe that many pages within an organisation have a similar “template” defining a common set of information to be provided and providing more or less stringent layout guidelines. For example most research institutions provide a set of pages for each on-going project, which may contain the name of the project, its objectives, leader and staff, amount of funding, experimental results, demonstrations, publications *et cetera*. A sizeable proportion of project pages will follow a standard form, without necessarily being identical in terms of layout or hyperlink structure.

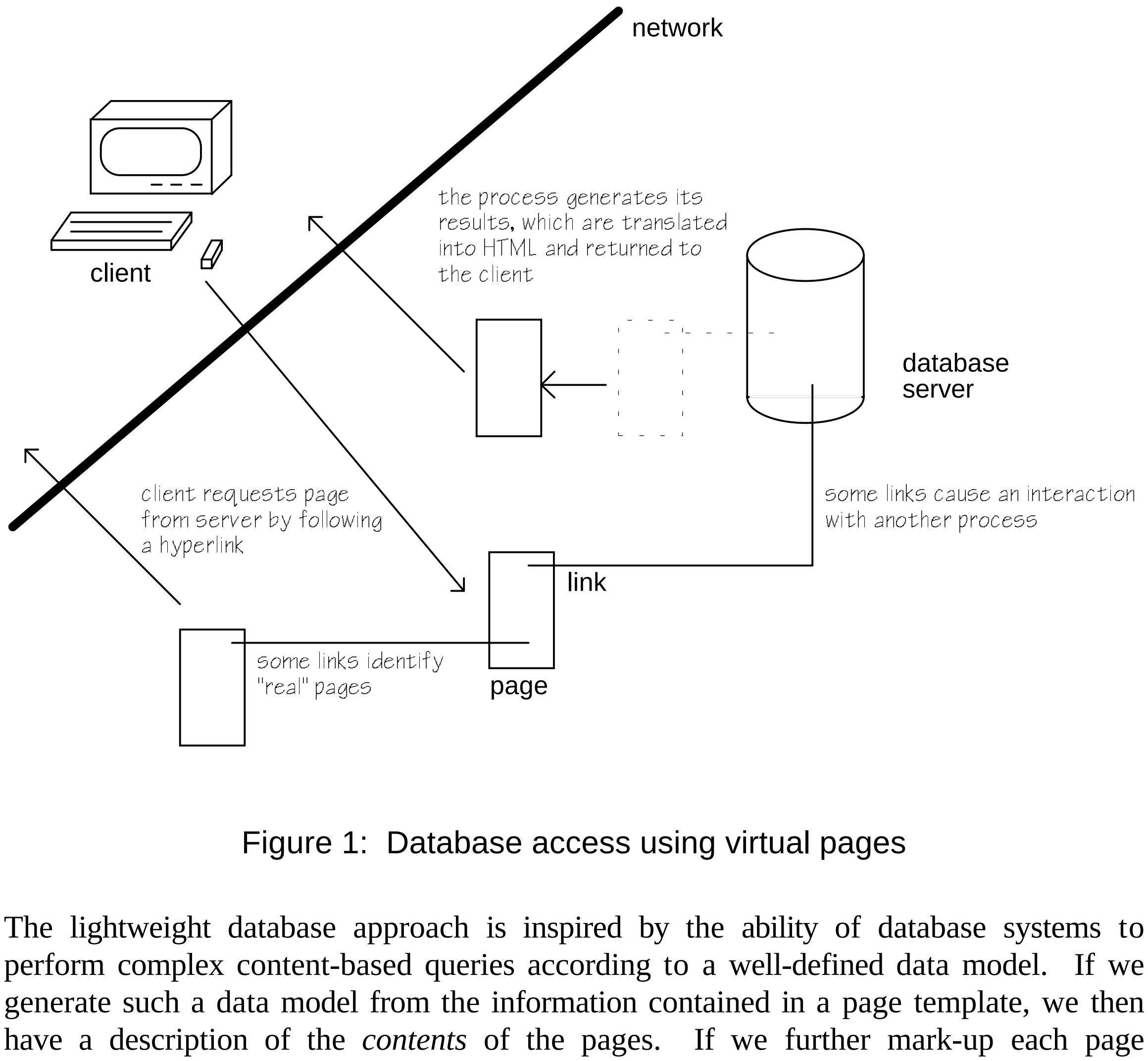
network
client
the process generates its results, which are translated into HTML and returned to the client
database server
client requests page from server by following a hyperlink
some links cause an interaction with another process
link
some links identify "real" pages
page

Figure 1: Database access using virtual pages

The lightweight database approach is inspired by the ability of database systems to perform complex content-based queries according to a well-defined data model. If we generate such a data model from the information contained in a page template, we then have a description of the *contents* of the pages. If we further mark-up each page according to this model and match parts of the page with corresponding parts of the model, we have a representation of the information which may be *browsed* using WWW clients or *queried* using a database-like query engine. (Another way of looking at this is that the extended HTML pages are simply the physical storage model for the database, which may use a different representation than the conceptual model.) We may also include relationships between entities according to the data model, which may or may not be reflected in the hyperlink structure of the pages.

A lightweight database thus provides an alternative view onto a collection of information, a view based on its semantic content rather than its presentation. The two views are largely orthogonal, so that a page designer may develop pages with few presentational constraints. In particular, it is important to realise that the lightweight database's entities are connected using relationships, *not* hyperlinks: the hyperlink structure of a set of pages can be different from the relationship structure of the database, and typically the former will be significantly richer than the latter.

Extended Mark-up

In this section we give a brief overview of our extensions to HTML for handling semantic mark-up for lightweight databases. A more complete discussion including motivations and details of a previous attempt may be found in [5], which also contains an informal DTD of the extended mark-up.

Creating a lightweight database is a three-stage process: agree the underlying conceptual data model, create the pages, and mark-up the pages according to the model. Many organisations already describe their structure according to a data model; for others, and for other sorts of pages, it may be necessary to decide exactly what information is to be stored. Once this has been decided and formalised the pages may be put together and linked using whatever hyperlink structure is required before the lightweight database mark-up is added.

The mark-up consists of three extra HTML elements:

- `<ent>...</ent>` identifying an entity;
- `<attr>...</attr>` identifying the value of an attribute within an entity; and
- `<rel>...</rel>` denoting a relationship between entities.

These are HTML-style elements with no presentational content, which should be (and are) ignored by browsers. Each element can take a series of attributes supplying important information about the contents of the element, matching it against parts of the underlying conceptual model.

An Example Lightweight Database

To illustrate the use of the notation, we shall mark-up a page which describes a research project. We begin by defining a suitable conceptual model for a project. Since for many organisations (including our own) projects are very similar, it is often possible to derive a canonical model which encapsulates the core information: this still allows individual projects to include extra information on their pages if desired. The model is shown in figure 2. Each entity class has a *primary key* (indicated by underlining in the diagram) which identifies each instance uniquely within a class.

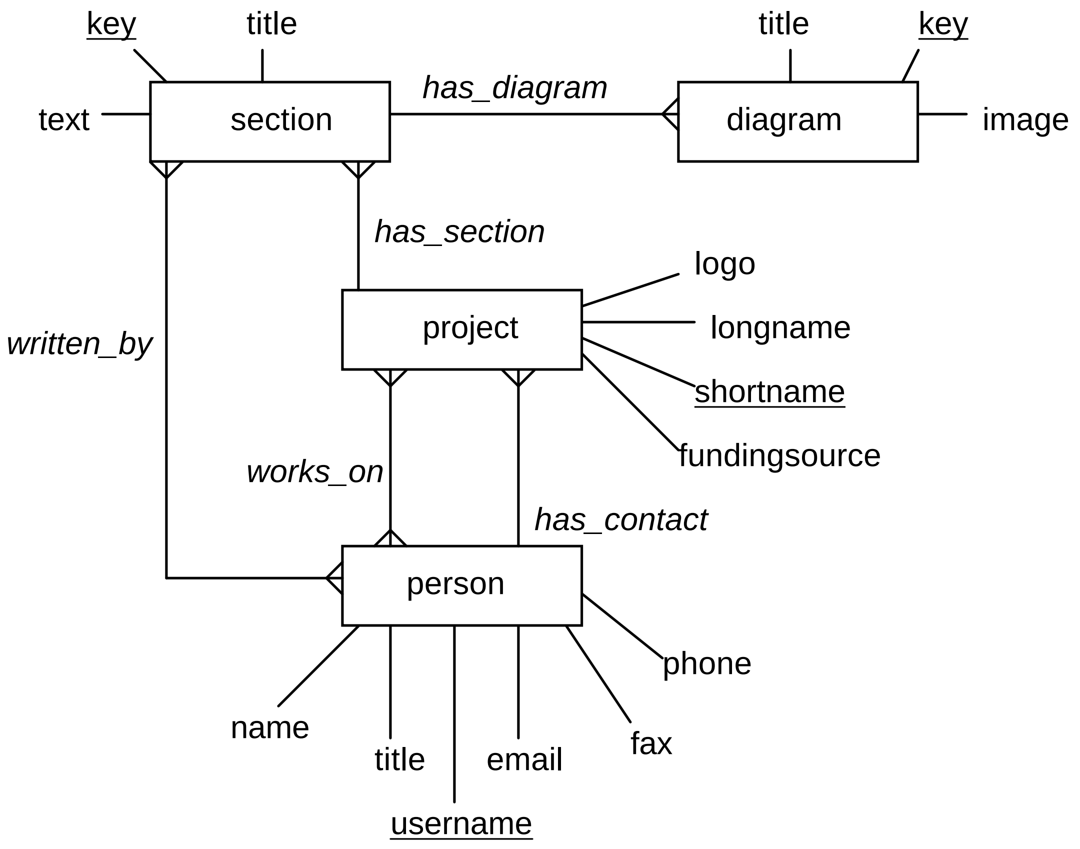

Figure 2: Conceptual model for project descriptions

Each project is represented as a single entity of the *project* class. This class has attributes of long and short names, logo *et cetera*. We mark-up the top-level project entity as follows:

```
<ENT KEY=mips CLASS=project>
<H1>
<ATTR NAME=logo><IMG SRC="MIPS.gif"></ATTR>
The
<ATTR NAME=shortname>MIPS</ATTR>
Project
</H1>
...
</ENT>
```

The `ENT` element provides the primary key value and entity class of the information. Significant pieces of information are captured within `ATTR` elements, which identify attributes by name. The page itself contains extra formatting information, such as the use of `H1` for the heading, but this is not significant (from a database point of view) and lies outside the scope of the identified attributes.

Entities are linked using relations. In our example, the MIPS project is linked to a particular person under the *has_contact* relation. We may encode this as:

```
<ENT KEY=mips CLASS=project>
...
<REL NAME="has_contact" CLASS=person KEY=mdw
HREF="/people/mdw.html">
<A HREF="/people/mdw.html">Michael Wilson</A>
</REL>
...
</ENT>
```

The `REL` element relates the containing entity to another entity identified by key value and entity class under a named relationship. The `HREF` attribute provides location information, indicating the URL at which the target entity may be found. This simplifies searching, as a search engine may move directly to the target of the relationship. We regard this as a useful navigational hint, not as an essential part of the mark-up – in particular we do not *require* that the entity exists at exactly the given URL.

Notice that the `REL` element contains a denotation of the relationship, which in this case is a hyperlink. One may form relationships between entities without using hyperlinks, or hyperlinks without invoking a relationship (in the database sense). This is especially useful when we want to provide information in the database which is not directly accessible by a human browser: a relationship without a corresponding hyperlink is effectively invisible, although a database-aware client could still traverse the relationship to acquire the information.

We may also wish to relate entities which are stored in the same document but which are logically distinct. For example, the MIPS project's introduction is a *section* entity presented along with the project's top page. This relationship may be encoded by nesting an entity element directly within a relationship element:

```
<ENT KEY=mips CLASS=project>
...
<REL NAME="has_introduction">
<ENT KEY="mips_introduction" CLASS=section>
Most consumers are...
</ENT>
</REL>
...
</ENT>
```

Again, the database structure of the information is separate from the way it is presented in terms of in-line inclusion and hyperlinks. One may see the information as being stored in a way which is most convenient for the most common mode of use (browsing) whilst remaining accessible to other modes (database queries).

4. Automatically Accessing Information

A lightweight database is not an end in itself, of course: it is simply a means whereby extra value can be added to WWW pages, enabling more complex applications to be implemented within the framework. It may be combined very effectively with other approaches to searching and automation. In this section we describe three illustrative example applications which we have investigated, and examine ways in which further progress may be made in automating WWW.

a. Querying and Virtual Databases

The first use of the mark-up is to allow better searching for information within WWW pages by exporting their underlying information content. One way to improve searching is to move more towards embedded RDBMSs, allowing queries to be presented directly to a search engine. An alternative is to provide a query engine which understands the lightweight database notation and can query pages directly. We have implemented a simple template-driven query engine which can perform arbitrary queries on a lightweight database, returning the results as an HTML page.

One advantage of this approach is that the information-carrying elements of pages are published along with the data. This makes client-side database manipulation possible[4], as the *database* is made available rather than just its *contents* (as is the case with an embedded RDBMS). We are currently investigating ways of making conceptual models available to clients (possibly through a well-known page at all sites) to facilitate the construction of complex queries and conversion of data models across sites.

Furthermore the relationships in a lightweight database use URLs as hints as to where to find the destination entity. This means that sites agreeing a data model for (parts of) their information may create relationships to each others' entities. This exploits the location-transparency of URLs to create a large *virtual lightweight database* out of several individual servers. One may represent a large database in a scalable manner, adding more servers as required, and the database survives in part if any component server fails.

b. Indexing

Current approaches to index creation using “robots” – autonomous dæmons which scavenge for documents on the web – are extremely network-unfriendly and can be inaccurate, often mis-classifying or omitting pages. Their reliance on keywords as a guide to content can also cause problems in complex domains.

One may make the process less hit-and-miss by having high-level conceptual descriptions of a site's contents available at a known point. An index generator may then check a single document to determine whether a site holds information in which it is interested, rather than scan a complete web looking for keywords.

A more complex solution involves migrating WWW away from a simple client/server model, introducing more complex distributed systems ideas. This work is beyond the scope of the current paper, but we are considering several extensions to WWW to allow sites to exchange update information so as to maintain semi-automatically indexes and other cross-references.

c. Generating Printed Copy

The original motivation for our work on lightweight databases was the need to generate “flat” printed copies of documents from their corresponding web pages with a complex hyperlink structure. This is non-trivial because one does not necessarily want to include data from all hyperlinks, and because parts of the hypertext presentation may not be suitable for rendering on paper – hypertext and flat text are fundamentally different media.

By defining a conceptual model of documents and marking-up their pages accordingly we are able to generate a “flattened” document as the result of a lightweight database query. This means that the same representation may be used for both printed and hypertext formats: the hypertext version is the master copy, but we may generate the printed version if required.

Automation and Programming

Widespread use of WWW to disseminate information makes it an ideal candidate infrastructure for a wide range of distributed applications. An application designer could use existing WWW servers and technology to support an arbitrary application, retrieving and manipulating objects identified by URLs or extracted from lightweight databases. This goes beyond automating WWW and instead imports the web as an integral part of an application framework.

Much of the infrastructure already exists, as the hypertext transfer protocol (HTTP)[1] used in WWW is far more general than current browsers would indicate. As well as object acquisition it supports arbitrary object types admitting user-defined methods, and has sufficient power to act as the transport mechanism for a lightweight distributed object system.

Existing programming methodologies are probably inadequate to deal with creating large-scale, general-purpose applications distributed across Internet. As part of an EPSRC-funded collaborative research project called *TallShiP* we are currently investigating the use of high-level data sharing in parallel and distributed programming. TallShiP represents applications using *shared abstract data types* (SADTs) which encapsulate common large-scale data structures in a scalable way, using advanced language techniques and transformations to map these abstractions onto underlying resources. One example SADT is a graph being used to represent a set of hyperlinked pages: one may then create WWW-like applications using a high-level model, allowing the compilation system to generate the necessary client/server code.

5. Conclusion

In this paper we have discussed the need for improved automation of World Wide Web in order to improve its usability as a means of disseminating information. We have described our experiments in creating lightweight databases by including into their mark-up the semantics of documents. This allows general database structures to be captured within almost-standard HTML documents, providing a structured view onto pages which may be exploited by suitable database engines.

Amongst the applications of this technique are those to improve the searching and indexing capabilities of WWW, the creation of virtual databases spanning servers, and the generation of printed documents from sets of hyperlinked pages.

We have also suggested a need for improved architectural models and programming methods within WWW in order to manage the complexities of applications using information distributed Internet-wide.

6. References

[1] Tim Berners-Lee, "*Hypertext transfer protocol: a stateless search, retrieve and manipulation protocol,*" Draft RFC, (1993).

[2] Tim Berners-Lee, "*Uniform resource locators: a uniform syntax for the expression of names and address of objects on the network,*" Draft RFC, (1993).

[3] Tim Berners-Lee and Daniel Connolly, "*Hypertext markup language specification - 2.0,*" IETF HTML Working Group (1994).

[4] Paul De Bra and Reiner Post, "*Information retrieval in the World-Wide Web: making client-based searching feasible,*" in Proceedings of the 2nd International World Wide Web Conference (1994).

[5] Simon Dobson and Victoria Burrill, "*Lightweight data mark-up,*" WWW/04/94, Computing and Information Systems Department, Rutherford Appleton Laboratory (1994).

[6] Simon Dobson and Victoria Burrill, "*Lightweight databases,*" to be presented at the 3rd International World Wide Web Conference, Darmstadt (April 1995). Also available as WWW/09/94, Computing and Information Systems Department, Rutherford Appleton Laboratory (1994).

[7] Carlos A. Varela and Caroline C. Hayes, "*Zelig: schema-based generation of soft WWW database applications,*" in Proceedings of the 2nd International World Wide Web Conference (1994).