DRAL

Daresbury Laboratory
Rutherford Appleton Laboratory

Rutherford Appleton Laboratory
Informatics Department

S.A. Dobson, V.A. Burrill

Preliminary Results from the Database Mark-up of Hyperdocuments

4 November, 1994
WWW/03/94

This document describes some early results which we have obtained in marking-up World Wide Web (WWW) documents in a novel style which attempts to treat hyperdocuments as a database. We have created a small "weblet" of sample pages to which we have added the capability to generate printable documents direct from the hypertext. This capability is provided without modifying any existing code, and requires only minimal alteration of the hyperdocuments.

We intend this experiment to serve as a proof of concept, not as a finished system. There are several problems still outstanding – both foundational and with the current implementation. We hope that our preliminary results will provide pointers for potential future developments.

Database Mark-up

Entities and Relationships

A database is defined according to some underlying data model, used expressed in terms of an entity-relationship (ER) diagram. The diagram identifies the entities which exist in the universe of discourse of the database, their attributes, and the relationships which hold between them.

Each entity instance has a combination of attributes which allow it to be uniquely identified within its entity class: this is called its *key* value. Keys are important in that they allow relationships to be defined: an instance of one entity class is related to one or more instances of another entity class, where the instances are identified by their keys.

Anchors in HTML

WWW uses the Hypertext Mark-up Language[1] (HTML) to describe hypertext documents. HTML is an SGML-compliant language which defines the logical structure of a hyperdocument. Browser software may render the different logical elements as it chooses, according to user preferences.

An *anchor* is a tag with a hyperdocument identifying a connection within the hyperstructure. There are two kinds of anchor. *Link* anchors define a link to another

hyperdocument, which the user may traverse by selecting the link (usually with the mouse). The link is identified in the source document by a small denotation – a piece of text, icon or image – which gives a cue to the user as to the effect of selecting and traversing the link. *Expand-in-place* anchors identify a part of the source document by another name. The contents of the anchor are displayed directly in the source document, often without any indication that there is an anchor present. It is possible, however, to specify an expand-in-place anchor as the target of a link.

In HTML all links are specified by providing a Uniform Resource Locator[1] (URL) for the target document. Expand-in-place anchors have URLs derived from the URL of their containing document.

Within HTML a link anchor is defined using the following syntax:

```
<A HREF="--url--">--denotation--</A>
```

where *--url--* denotes the URL of the target document and *--denotation--* denotes the text (or other element) which will appear in the source document to denote the link. An expand-in-place anchor uses a similar syntax:

```
<A NAME="--name--">--elements--</A>
```

where *--name--* is the name of the anchor and *--elements--* is the contents of the anchor which is to be inserted.

A link may also have a type indicating a relationship between source and target objects[1]. A link is given a type using an additional keyword in the HTML, so the tag

```
<A REL=--rel-- HREF="--url--">--denotation--</A>
```

is a link with type *--rel--*. Anchors with explicit type have the default type of void.

Isomorphism

For database mark-up we make the following identification. An entity in the data model gives rise to some hyperdocument identified by a URL: either a document or an expand-in-place anchor within one. A relationship gives rise to a link between the appropriate URL-identified objects, with the link being typed using the same name as the relationship.

An entity must have its key value encoded into its identity – URL – in some way. For entities held in documents, this must be the file name; for entities held in expand-in-place anchors it may include the name of the anchor.

Since each entity is represented by exactly one object, we may make the following observation: there is no need for the visual form of the hyperdocument store (the

---

[1]Currently HTML allows type only on links, not expand-in-place anchors. We view this as a mistake in the specification. For the remainder of this paper we assume that a type may be assigned to any anchor. In practice this is supported by all servers and browsers.

hypermodel) to conform to the arrangement of the underlying data model. This is because the entities and relationships are encoded in the hypermodel in a manner which is independent of the arrangement of the anchors. In particular the link structure of the web may be (and in general will be) richer than that of the data model: but the extra links will be untyped and will thus be invisible to any system expressed purely in data model terms.

Thus one may choose an appropriate visual representation for a web – based on the usual principles of usability, and providing an attractive and interesting presentation of the information – secure in the knowledge that the underlying data model is not compromised by any "prettiness" which is added for the sake of presentation. Data and hypermodels are *not* the same, but with care they may be made isomorphic to one another.

Transformation System

Having identified entities and relationships from the data model within the hypermodel, we may make use of the information to provide database-inspired functionality. We demonstrate this by providing a transformation system which can extract data from a hyperdocument structure according to criteria from the data model, and re-format the data into a new hyperdocument.

Templates

The new data is generated as HTML – exactly like the source – so that the results of a "query" is a new hyperdocument. The resulting hyperdocument is defined by a pro forma or template. A template is an HTML document expanded with two new kinds of anchors: object insertions and denotational insertions. These extra tags are used to control the insertion of data extracted from the hypermodel by the transformation system and are never included in a generated hyperdocument (which is thus always "pure" HTML).

An insertion specifies an entity in terms of a relationship and/or name (key value). The object insertion tag tells the transformation system to insert the full contents of the identified object into the resulting document. A denotational insertion tag is similar but inserts the denotation of a link rather than its target object. For an expand-in-place anchor the two kinds of insertion are identical; for a link, a denotational insertion effectively "flattens" the hyperdocument by removing the linked data.

If the requested object cannot be found in the hyperdocument structure, the insertions are not expanded and simply disappear from the resulting hyperdocument. We allow pre-and post-formatting elements to be included within the insertion tags, which are only expanded if the tag is expanded. This means that one may (for example) specify a header to be included with a reference list, but neither the header nor the list will appear if there are no references available.

Transformation

The transformation application scans a hyperdocument structure to extract all the objects within it, and extracts objects specified by a template to generate a new hyperdocument. It is composed of three phases:

- an *HTML parser* which tokenises an HTML document;
- an *objectifier* which recognises anchors and extracts objects, recursively traversing any links encountered; and
- an *expander* which fills-out a template and re-generates HTML from it.

The objectifier is the major part of the system. It recognises objects and stored them in an internal table. It traverses any links which have a non-void type and recursively extracts and objects it finds.

The transformation system is implemented in around 500 lines of the Caml Light dialect of ML, and took a single day to write.

Integration into the Web

In order to perform a transformation on a document the user must be able to invoke the transformation application through the web, supplying both the initial hyperdocument structure and the template to be filled out.

We have implemented a small collection of shell scripts which call the transformation application with the appropriate parameters. There is a shell script for each kind of template – each allowable query, in database terms – and these scripts are the only way in which the transformation system may be invoked by users.

The scripts are placed into the */cgi-bin* directory of the WWW server. A link is then inserted into the hyperdocument to be printed, identifying the appropriate script and the name of the base hyperdocument to be transformed. The server will call the script automatically, passing it the hyperdocument's name as a parameter. The script in turn calls the transformation system and returns its result to the user as a new hyperdocument.

Examples

We have implemented an isolated weblet on the departmental web server to test the transformation system. The weblet is a severely cut-down version of the standard departmental structure, derived from the public version and marked-up as necessary. It took around a day to generate the weblet from scratch, which is an indication of the small amount of mark-up necessary.

The weblet may be accessed from a browser using the URL

```
http://web.inf.rl.ac.uk/Odds/paper/
```

(it is not accessible directly from public parts of the web) and follows the usual departmental conventions in terms of navigational icons *et cetera*. Some pages contain an additional icon which invokes the transformation system to produce a printable (non-hypertext) version of the document.

Project Hand-outs

The first example is of project hand-outs, to be given to companies or prospective partners wanting a brief summary of existing research efforts and a contact point for further enquiries. Although the web pages describing projects were (in the main) derived from the previously-existing hand-outs, the needs of hyperdocuments have led to many being expanded. In particular some information is contained on other pages accessed *via* a link, and several projects contain examples and diagrams which were not in the printed version. One cannot directly print a hand-out from a document which contains hyperlinks, as the browser will not know which links should be followed, expanded or ignored.

The transformation system allows a template to be defined for a project hand-out. This defines the layout of the hand-out together with the objects which should be extracted (if possible) from the hyperdocument structure and inserted. Selecting the print icon will generate a document without any hyperlinks which adheres to the template and is suitable for printing. All project hand-outs will follow the form defined in the template (although there may be variations, for example between on-going and completed projects), regardless of the exact hyperdocument structure from which they are derived. This means that project descriptions may be stored in the web in any desired form without compromising the ability to generate hand-outs on demand.

Contact Sheets and Biographies

Each person in the department has a personal entry in the web. This includes at least the person's name, telephone and fax numbers, e-mail and postal addresses. A person may optionally supply a biography and list of references for inclusion in the web.

A person's contact details may be printed, generating a contact sheet. If a person has provided a extra information, one may generate a sheet containing the person's biography, reference list and contact details. Both these sheets are defined using templates, so may be standardised across the department.

Conclusions and Further Work

We have described a small experiment which allows a hyperdocument to be treated in some measure like a relational database, and have demonstrated the uses of this approach through a capability to generate alternative forms of hyperdocuments suitable for printing as plain text. The work involved in making these facilities available is very small: a small amount of mark-up for each page plus the generation of appropriate templates.

There are still outstanding issues, however. The most important, perhaps, is better to understand the implications of this style of mark-up. It is not clear, for example, whether

we should identify entities in the data model with expand-in-place anchors only. These are foundational questions which affect our ability to perform generic database operations with maximum flexibility.

The transformation system, as it stands, is not the best piece of code. It would benefit from re-writing, especially after the above foundational questions have been resolved. It is also currently very memory-hungry as it eagerly creates an image of a hyperdocument structure in memory. This could be avoided by supplying the transformation application with more details of the underlying data model. There are also many additional database operations which might usefully be provided, along with extra template tags.

Finally there is as yet no formal description of the data model underlying the departmental web – and, more generally, the proposed unified DRAL web. Once this exists it will be possible to define the form and content of the printed documents which must be generated, which in turn will provide a better view of the necessary capabilities and scope of a hyperdocument transformation system.

References

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

[1] Tim Berners-Lee and Daniel Connolly, "*Hypertext markup language: a representation of textual information and metainformation for retrieval and interchange,*" Draft RFC, (1993).