Building Virtual Documents upon Distributed Databases

Anne-Marie Vercoustre, Jon Dell'Oro

CSIRO, Division of Information technology
723 Swanston Street, Carlton,
Victoria, 3053, Australia

Anne-Marie.Vercoustre@dit.csiro.au
Jon.Dell'Oro@dit.csiro.au

Abstract

This paper explores the issue of publishing information under the form of Virtual Documents that include data and fragments of documents from remote sources, especially from databases and SGML Documents databases. Because they are dynamically generated, Virtual Documents are automatically updated when the source information changes, and they allow for sharing, reusing and adapting information for various contexts. The paper focuses on implementation of virtual documents from the authoring and architecture perspective.

1. Introduction

Much corporate information now resides in electronic documents, and Document Management has replaced data management -- a focus of computing research over the last two years. The WWW has gained widespread popularity as a means to access this information, both by private use within an organisation as well as for public availability. The Internet and the Web provide the basic mechanisms for Information Discovery in widely distributed heterogeneous networks, but more advanced support is necessary for finding and providing the right information for the right people.

Information Discovery, or more generally, Resource Discovery, commonly refers to the issues of efficiently discovering, retrieving and managing relevant resources from networks. The Yarra project[1] addresses more specifically the issues involved in making these resources available over a network for further retrieving and usage, i.e. both the documents containing the information and the metadata describing these documents and where to find them. When coming to a very large collection of dynamic documents, such as on the WWW or even private corporate nets, the main issues for publishing information are:

- The automatic updating of the information contained within documents.
- The configurability of the documents according to user's needs, profiles or time line.
- The sharing and reuse of information through various documents.
- The automation and flexibility in providing meta-descriptions to deal with various and evolving required descriptions.

Publishing information on large networks is still mostly done by writing down the information into static HTML pages, making it difficult to update, share, process and reuse information. Another practice is to dynamically generate HTML pages by running scripts or programs on the web server. When the server is a gateway to a database the displayed information reflect the current state of the database and is always up-to-date. However, in this approach, there is no easy way of building other virtual pages using the same information from the database.

[1] The Yarra Project is a Resource Discovery Project undertaken jointly between the CSIRO Division of Information Technology and the DSTC. It employs metadata to describe resources and stores this information in a distributed database.

This paper explores the issue of publishing information in the form of *Virtual Documents* that include data and fragments of documents from remote sources, especially from databases and SGML document databases. Virtual Documents can be designed on a different server than the one where the initial sources of information are located, and could be instantiated on the client side. It is an extension to the current practice of Virtual Documents which are dynamically generated documents by a server or an application. ????????
This paper also gives requirements for a design language for building Virtual Documents. A Virtual Document description will include SGML-based specifications of the logical structures with (FOOTNOTE) OQL-like queries for retrieving content parts from a database. The Virtual Document description will therefore provide a *constructive* specification of the document.

2. Resource Discovery and Publishing

A Resource Discovery System is a framework which allows users to efficiently discover and retrieve relevant resources from a network. The Yarra project addresses the issues involved in making these resources available over a network, in particular:

- Creating (publishing) the original electronic resources.
- The construction, maintenance, and distribution of resource descriptions (metadata, indexes).
- Using the resource descriptions (metadata) to answer user queries and retrieve the selected resources.
- The distribution of copies of resource.

This paper focuses on the first objective, i.e. the publishing of the original resources in a flexible way from distributed sources of primary information.

For many web publishers today, information is published in two common ways: either information is authored into static HTML pages, or the web pages are dynamically produced by running scripts or programs on the web server.

In the first case, the information itself is described by the HTML document, which makes it difficult to update, share, process and reuse information. Hypertexts and the WWW have long been regarded as a good way of sharing and reusing information, since links allow references to existing pages without duplication. This approach encourages the browsing paradigm of accessing information, which may be time consuming and prevent the reader from gaining an overview of several pieces of documents. Some earlier Hypertexts, such as Guide [Bro89] were offering an include mechanism for expanding links within the current page, but this required a careful taxonomy of the overall hypertext, which obviously the World Wide Web cannot support.

Moreover, Web pages do not offer the appropriate granularity for inclusion, since they may contain different pieces of information intermixed with presentation elements that are not desirable in another context. (WHAT?)While HTML was initially designed for describing the organisation of the information and its semantic content (in the same way as SGML), it is now mostly used as a language for displaying information and is very inappropriate to identify and access specific parts on a semantic level. It appears that information managing approaches have been moving from a very rigid database approach dealing with very structured data to a world of flat texts supposed to represent any kind of information. Using more structured documents (eg. SGML) can benefit a lot from database support and especially so when documents are regarded as sets of information objects.

The publication of information on the Web should be the dynamic creation of documents from various sources of information which can be assembled in various ways. The next section will define Virtual Documents and review the current approaches in creating dynamic documents for the Web.

3. Dynamic and Virtual Documents

3.1 What is a Virtual Document?

While there are thousand of thousands of static pages on the Web, there are an increasing amount of dynamically generated pages. The simplest case of a dynamically generated page is an HTML document that is constructed in response to a search query. All search engines on the Web return such pages displaying a list of answers under the form of links to the actual documents and more or less information about the content (Title, abstract or extract of sentences, URL). Although such documents have a persistent name[2], and the content is virtual (since it can change over time), it is questionable whether it can be called a virtual document or not.

Virtual Documents have been defined [Gru94] as hypermedia documents that are *generated on demand*, in response to user (reader) input. This general definition seems at first to apply to the previous example, until we wonder whether those pages are actually "documents" or whether they just look like documents. The question of "what is a document?" [Norw96] is a difficult one, but a first definition could be that a document has two main functions:

- to communicate information
- to assemble and organise this information to make it knowledge clear and useable for the reader.

Those pages generated by search engines only assemble information which may or may be not relevant to the user, belong to quite different domains, or appear in several copies. So we are interested in the authoring and construction of virtual documents which are intended to assemble and organise information for supporting specific tasks or decision process. We would rather define Virtual Documents would as hypermedia documents whose structure and content can be easily adapted to the user task or need. They can therefore be regarded as a flexible mean to reuse information in a variety of contexts

3.2 Some examples of Virtual Documents

Virtual Documents are well-suited to present up-to-date information in domain where information is changing frequently, or to adapt themselves to the reader's communication needs.
The WWW offers many good examples of virtual documents:

- Virtual Conference lists [Conf] that display conferences events sorted by months, call for paper, date of events, etc...

- Many Interfaces to databases thought WWW gateway have been developed. They offer two different functions: database querying and updating through form-based interfaces, or database browsing by generating pages with embedded links. Example of the latter is the O2 AUTO demonstration [O2demo]

- Peba-II [Mil096] is a natural language generation system which dynamically produces hypertext description or comparisons of animals, based on a knowledge base of animals and a discourse and text planning component.

- DME [Gru96] is Question-Answer System that generates domain based-explanations with embedded links for follow-up questions.

These virtual documents are actually rather advanced examples, especially the two last ones which

[2] Here's an example of a page which has a persistent name, yet the content is virtual:
http://altavista.digital.com/cgi-bin/query?pg=q&what=web&fmt=.&q=Virtual+and+Documents)

involve natural language generation. Peba II, DME, and the O2 Web browser all use a domain model for generating pages with embedded links. The text itself is generated by the application, using techniques of varying degrees of sophistication, ranging from specific programs to generic programs which use hierarchical object classes, to natural language generation using text planning.

An instance of these virtual documents has a persistent context independent identifier (an unique URL) that allows to reference or reuse the document in another context. In fact the URL is simply the query that generates this instance of the virtual document.

The generation of the above virtual documents, however sophisticated, is completely determined by the foreseen application. The end user or another application accessing the same domain database cannot reshape it for another purpose. In this paper, we are mostly interested in describing virtual documents that allow the reuse of fragments of information which are distributed on remote servers or databases.

3.3 Architecture and processing

A typical application generating virtual documents like the above issues a call to a CGI program with specific parameters from an HTML document (either using a form directly encoded in a URL). The HTTP server invokes the CGI program (possibly through a gateway to the application). The result of the CGI program is usually an HTML document built by the application that is passed to the client through the HTTP server.

The broad common model is described by Figure 1.

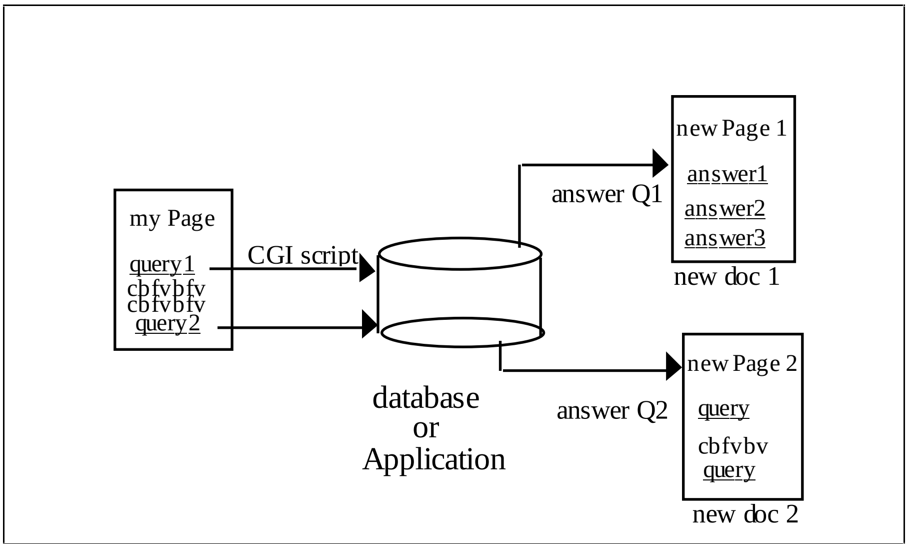

Figure1.

The Virtual Documents Doc 1 and Doc 2 are generated by the application and contain embedded links which may generate further call to CGI scripts if they are activated. In this model, called querying by browsing, the user is given more information by browsing through pages that are generated only if the user follow a particular path.

However, accessing the desired information can sometimes be very time consuming, and it can be difficult to get an overview of a sophisticated set of pages. The structure of the resultant pages are completely determined by the web programmer, and there is no provision for automatic generation of a abstract by combining various pieces of information that are retrieved by queries issued at various

level of browsing. ???
We want to take the reverse approach, ie. a document-driven approach, in which a virtual document would be defined that contains parts instantiated by queries to an external application or a database, like in Fig.2.

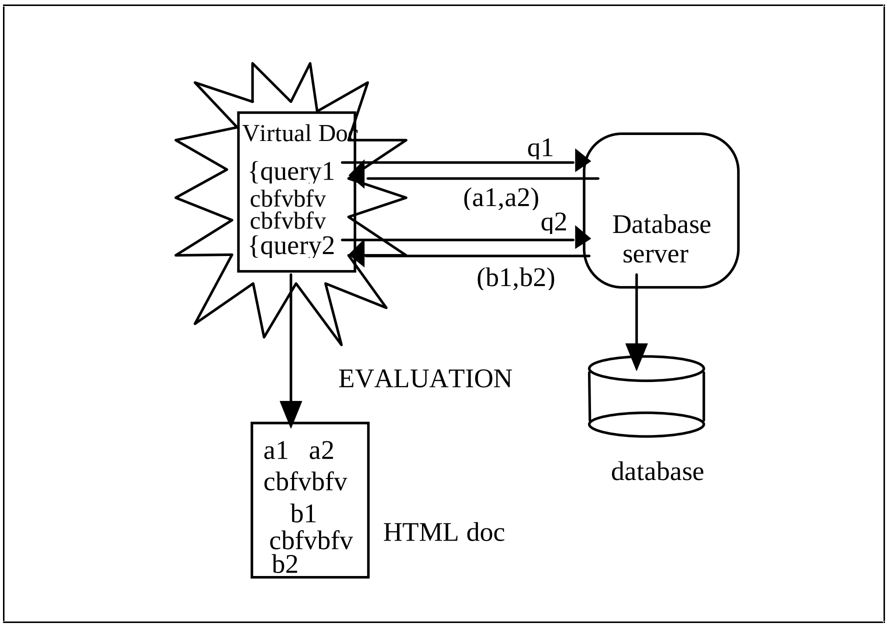

Figure 2.

(??? Should draw a page that looks a lot like the one from the SGML conf)

Fig.2 shows that all the queries are instantiated at the same time and their result is included into a document according to what is specified by the virtual document. The organisation of the information within the virtual documents can be then driven by the usage of the information, rather than by internal model of this information. It means also that information can be reconFigured independently of the database developers.

So 2 issues (which do not cover all aspects of virtual documents):
- virtual documents as document constructors (authoring)
- virtual documents that can reuse existing information for specific purpose, which implies not to be constraint by the initial model of the information source.

3.3. Current techniques for creating Virtual Documents on the Web

The model describe in Fig.2 requires as a minimum some mechanisms for including new elements within an existing document. It is surprising that HTML itself does not offer such a capability[3]. However, the Web already offers a range of mechanisms for creating Virtual Documents, the most used being through a CGI-script that creates a complete new HTML page. More specific techniques for including pieces of information within a static HTML page includes:

- an HTML document that contains <IMG> tag with an URL for selecting the images to be included into the document. The URL can specify a file name (.gif) or a request to a database or a

[3] Although the HTML spec itself does not allow includes from other pages per se, one of the early extensions to web servers was the notion of a server-side include, so that a web author could make a reference to a program and the server would run this program whenever the page is requested.

CGI script that return in formation (eg. the value of a counter). The same can be achieved using a form which returns an image. The W3C is currently working on an HTML extension for multimedia objects insertion within a document though a new tag called <OBJECT>, as a generalisation of the <IMG> tag [].

- an HTML document that contains parts to be included at loading time using the include facility on the server side.
- Frames are a Netscape extension that allows for displaying various sources of information in the same browsed page. It does not provide inclusion within the same document, rather inclusions of several documents within the same browser page in a bounded-box layout scheme.
- Java applets for inserting executable code.
- Advanced interfaces to databases, which we have explored in the examples given in Figure 3.

4. Experiences with distributed Virtual Documents

4.1 Motivation

The motivation for building several examples of virtual documents was to gain experiences in various techniques for implementing them, both at the authoring level and at the architecture level. The virtual documents we implemented are the following:

- A virtual organisation of bookmarks lists through a graphical presentation.
- A virtual document on the top of the O2-Web gateway.
- A conference list based on the PHP gateway to an SQL database.

In the following sections we will describe the motivation for each document, the architecture and techniques used as well as some conclusions.

4.2 Virtual Bookmark list

The dynamic bookmarks program is an special case of a virtual document. It provides an alternative presentation for a user's bookmarks; a number of frames is laid out in a grid, and each bookmark folder is placed into a separate frame which is resizable. It also uses a random image as a border.

This example is not a sophisticated virtual document because it doesn't filter the information in the bookmarks themselves, nor add more information to the bookmarks based on the folder title. However, the alternate presentation itself is a catalyst for the viewer to change their browsing patterns and get a more graphical and compact view of his bookmark-list.

The input for this program is the URL for a Netscape bookmarks HTML file. Anybody can test this program with their own bookmarks, if their bookmarks are available on the web (usually from the user's home page). The URL for testing is:
http://www.mel.dit.csiro.au/~delloro/dynamic_bk.HTML.

This virtual document as been implemented as a CGI-script that parse the Bookmark list and generates the various HTML files to be assembled in separate frames.

4.3 A Virtual Document upon the O2web gateway demonstration

The example is built using the Object-Oriented O2 database [O2], on the top of the O2Web on-line demonstration developed by the O2 company.
The original O2 **Auto Demonstration** allows for browsing through a database describing cars, parts of cars, price information, suppliers of the components, manufacturers, etc. It also allows for **logging** as one of the users and see, create and delete your orders. It is also possible to issue ad hoc

queries to the database using the OQL query language. OQL is the standard object-oriented query language, defined by the ODMG (Object Database Management Group). OQL is very similar to SQL but it allows complex object database to be queried and object methods to be invoked.
All the displayed pages are dynamically built by the O2Web server through appropriate *methods* which are invoked when the queries are issued.

For example the OQL query :

```
Select o
    from u in Users->users, o in u->orders
    where u->title = "Laura Mars" or
          u->title = "Julia Andrews"
```

returns all the orders by Laura Mars and Julia Andrews, with the following standard presentation:

O2Web

- 01/07/1995 (Julia Andrews, 953455
- 01/08/1995 (Julia Andrews, 953456)
- 01/09/1995 (Julia Andrews, 953457)
- 0/04/1996 (Julia Andrews, 953505)
- 01/07/1995 (Laura Mars, 953461)
- 01/08/1995 (Laura Mars, 953462)
- 01/09/1995 (Laura Mars, 953463)

Figure 3. Virtual Document designed by the O2 server

Different people may want to organise and display information in a specific way while any O2 application can only design one (possibly including parameters) HTML virtual documents model. Changing that model will require overloading the appropriate methods and cannot be personalised.

We wanted to build virtual documents can merge several levels of data in a document-oriented presentation rather than the browsing-oriented format defined by the database methods.

Our virtual document displays the list of orders by Laura Mars and Julia Andrew under the following format:

List of Orders by Laura and Julia

Laura Mars

- 01/07/1995
- 01/08/1995
- 01/09/1995

Create a new order

Julia Andrews

- 01/07/1995
- 01/08/1995
- 01/07/1995
- 0/04/1996

Create a new order

Figure 4. A remotely designed virtual document.

It is a virtual document since its content will update when the database is updated, but its format is completely defined outside the O2Web server.

This simple virtual document is generated by a program which create the static document parts (title, headers, lists) and issues the appropriate OQL queries, extract the interesting information from the resulting pages, and builds the lists with the corresponding content. As the Web page created by O2 in respond to the query:

```
Select o
     from u in Users->users, o in u->orders
     where u->title = "name"
```

has always the same format, it was possible to build a generic parser for extracting the interesting content of different pages and then merge them into a single document. We also added the appropriate buttons for each of the two customers to order new parts from its own overview, creating then a much more personalised environment.
This prototype was in some way a precursor to WIT [Trans96], a system developed by TransactNet, for building classes and objects that represent the content of Web pages which conform to a regular format. Classes and objects are generated in Java and can be used by external applications. In our case the external application would be the construction of a virtual document that contains the extracted information.

The virtual document could display deeper levels of information than the above example by using recursive queries.[4]

---

[4]The OQL language would allow for returning the appropriate result with the only following query:

```
(Select  tuple(u, o)
     from u in Users->users, o in u->orders)
```

but the O2Web server can’t manage it at the moment, so we have to issue two queries.

```
For each $i  in
      (select  i from i in Users)
      build-header($i.title)
      build-list(select o
                     from u in Users->users, o in u->orders
                      where u->title =   $i.title)
```

Figure 5. A virtual document with a recursive query

Unlike the query evaluation in Figure 1, where one query is evaluated when a link is activated, the evaluation of our virtual document triggers all the queries and synthesises a document which assembles the answers as designed.

4.4 A Conferences list

Our second example of a virtual document with its content built from a database is a list of conferences about Hypertexts, WWW, Electronic Documents, Advanced databases, Knowledge bases, etc. all subjects of interest for our research group. The conference lists aim to be user-friendly, displaying the lists with various sorting criteria , such as topics, countries, dates or calls for papers. Additionally, extra conferences may be added by anyone on the web using a form.
The URL for the conference list is http://www:8080/~vercous/Conferences-list.HTML

The data for this conference list has been stored in an SQL database called mSQL. This is a unix-based database that implements a useful subset of ANSI SQL. We first accessed the database using its associated gateway to the web, called w3-msql [???], which we were initially impressed with. The w3-msql program works by embedding SQL queries in HTML comments; when this file is requested, the web server calls w3-msql and the queries are replaced with the results. However, as we progressed, and wanted to produce more sophisticated pages, the w3-msql language soon proved to be lacking in power, with insufficient control of complex queries, and a lack of security features[5]. Then we moved to another product called PHP/FI [Ler96]. This is a rich, mature wrapper for web pages that includes access control, support for database connections, language statements, and string processing. Among the many useful features of PHP is the ability to function as a module to the unix web-server, Apache, thereby running in the same process space and performing operations very quickly.

Like w3-msql, PHP programs works by embedding SQL queries and control instructions in HTML comments. We ended with only two HTML documents (actually .phtml files as processed by the PHP server). The first document defines the various conferences lists while the second one is a form for adding or modifying a conference entry. The first document is very complex since it controls:

- the content of the document by issuing the appropriate queries according to the selected sorting criteria.
- the organisation of the document by creating the appropriate grouping of information.
- the presentation of the document by embedding the content into HTML tags.
- the user interface for switching from one list to one other, and from the browsing mode to the editing mode. It uses state information for adapting the interface and the display according to the context. For example the page background colour is changed to bisque when in editing mode to

[5] It should be noted that the w3-msql program that we were running at the time was an intentionally simple design. The author of the mSQL database has been working on a new version which addresses our problems; however, this was not available at the time we were developing our prototypes.

remind the user to be careful in editing the database.

It is not surprising then that the conference list that we started as a very simple virtual document ended to look like a relatively complex program, not better to some extend that other CGI-scripts . We will discuss that point in the next section.

4.5 Discussion

We will discuss our virtual documents from both the architecture point of view and the authoring point of view.

From the architecture point of view, our virtual documents ended as scripts running on a server and accessing distributed data, either from HTML pages (example 1 and 2) or straight from a database (example 3), according to the following broad picture (Figure. 6):

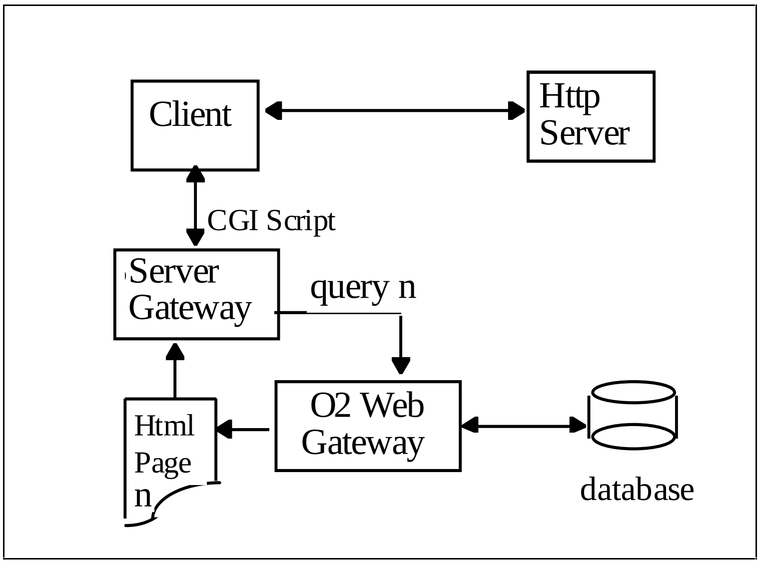

Figure 6. Architecture for the O2 example

The virtual bookmark list works with the same architecture except that the only query is an http request to the bookmark-list address (local file or URL). In the O2 example, the queries (2 in this specific example) are sent independently to the O2 Gateway and the result assembled into the virtual document. The architecture for the conference list is simpler since the PHP server handles a straight connection with the database (Figure.7):

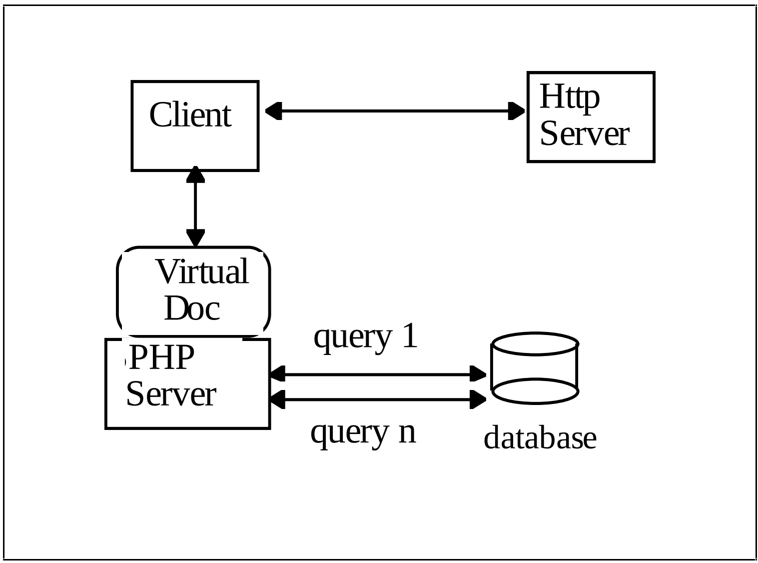

Figure. 7. Architecture using the PHP server

This a two-tiers architecture similar to the one described in [Dua96] for database applications on the Web using Java. Unlike JavaScript, PHP/FI is not browser-dependent and JavaScript is a client-side HTML-embedded language while PHP/FI is a server-side language

The main feature is that the virtual documents “constructors” and the data are not on the same server and can be designed by quite different persons. From the point of view of design and the reuse of information is a very important aspect. Anyone using a PhP server could design easily his own conference list from our conference database (providing we give reading access). Actually it would be much more convenient if there was such centralised conference database from which each research group could build a virtual one focusing on its research interest, instead of getting a fully general listing.

From the authoring point of view, using PHP was at first satisfactory when the document was an ordinary HTML documents with queries embedded into comments. It seemed much convenient than managing HTML and CGI files separately. Also it enforces the document-oriented design of the virtual document which was exactly what we were looking for. Unfortunately we started very soon to appreciate the utility and convenience of the control capabilities of the embedded programming language. Our virtual document then ended by looking like a relatively complex program with some print statements for outputting HTML tags, ie. much more as a CGI-script than an HTML document.

There are still some advantages:

- If you want to do simple things it is very easy and still looks like HTML.
- If you want to do complex things, you can put everything in the same file, with a lot of controls variables, and produce many different pages. These different pages share whatever it is possible to share and you have only one page to maintain.
- The integration with various databases is tackled by the PHP server, and you only need to know the query language syntax and your database schema.

5. Authoring Virtual Documents

We are interested in Virtual Documents that support the use, reuse and collaborative authoring of documents and especially documents that can combine information from distributed databases and fragments of existing SGML documents.

Our short examples above fulfill the requirement to access distributed sources of information but not the requirement for a suitable design support. We can consider two approaches to improve it: first to extend a language for documents description like SGML or HTML; the second one is to extend a document editor for supporting the retrieval, control and inclusion of external information within the document. The two approaches are complementary since the editor will need a language description for saving the virtual document.

5.1 Specification language

Our experience with PHP has resulted in a better understanding of the various components needed for the description language. The language would include:

- A query language for selecting and controlling the content elements.
- A mapping language for transforming the previous elements into document elements.
- A construction language with control capability for building the logical structure of the document; this language would include for example constructors for elements, lists, or fixed structures.
- A presentation language.

The language interpretation process can be described as following:

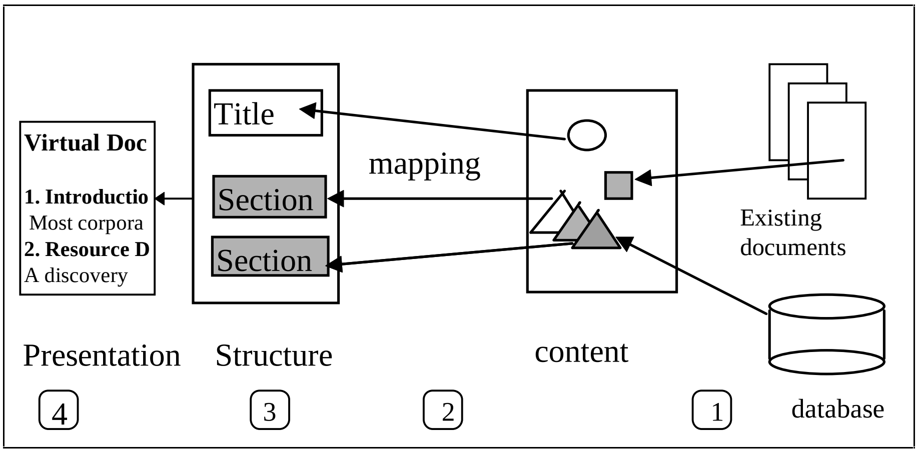

Figure 8. Document construction

1. The query language can specify which pieces of information have to be included into the document. It can be data from a database or fragments of existing documents.

2. The mapping language specify the transformation of the information objects into document objects. For example if the virtual document is an SGML document that includes fragments of other SGML documents, a mapping must be defined between SGML document elements to be included and SGML elements in the virtual documents, according to their DTD (Document Type Definition). This mapping could be defined using a middleware data model using labelled trees as proposed in [ACM96]

3. The construction of the logical structure of the document must include *operators* for building complex document elements like lists or aggregates. It must also include control capabilities for adapting the document structure to the environment (state variables or environment variable) and the actual content. Also it defines the static part of the document. The major difference with an SGML DTD is that the virtual document is a *constructive* description of the document and not a generic description for a class of document. However the description can take advantage of a DTD definition

for not redefining all the intermediary structures or common elements.

4. The presentation description specify how to format and display the document. It also includes control instruction. If the virtual document is an HTML document this part will not be distinct from the structure definition since both would be determined by the actual tags. In the case of SGML documents, the presentation can be generic to a DTD and specified for each SGML editor/browser in use.

If we consider again the example given by Figure.5, the syntax could look like:

```
for O2-demo use Article;                                              -- the name of a DTD

title =”List of Orders by Customers”;                                 -- a static title
section_l = (mapcar (Lambda x, section(section_title(x.u), item_l([x.o]))))
             (select tuple(u,o) from u in users, o in u->orders))
```

The query retrieved a list of tuples (customer, list of orders) which have to be mapped into a list of sections. Each section will have a section_title (the name of the customer) and a body which is a list of items. The list of orders and the item_l have also to be mapped, possibly using an external function. Currently all mapping are done at the lower level, eg. text.

We think that the International Standard for Document Semantic and Specification language, DSSSL [ISO/IEC96], could be a good candidate for defining Virtual Documents. Indeed DSSSL, defines the syntax, semantic and processing model of two languages for the specification of SGML documents processing:

1. The transformation language for transforming SGML documents marked-up in accordance with one or more DTD into other SGML documents marked up with another DTD. It provides access to, and control of, all possible marked-up information in an document through the Query Language component of DSSSL.

2. The style language that define the presentation of a document.

DSSSL transformation language defines transformations from groves into groves. A *grove* is a set of nodes connected into a graph (a more general structure than a tree). DSSSL includes an expression language to create and manipulate objects. The DSSSL expression language is based on the Schema Programming language (a functional language like Lisp). It is not surprising then that the previous example uses a Lisp-like syntax as we found it convenient for defining such mapping and constructions.

5.2 Extending an Editor

It is both possible and preferable to author Virtual Documents by using advanced facilities offered by document editors. We have been considering the extension of existing editors for including scripting language, by looking at Soda [Rat94], Grif [Qui94] as two of the most advanced SGML documents builders.

Soda

SoDa is an automated document generator built on the top of FrameMaker. It provides the capability of combining input from sources to create a new Frame document. It allows inclusion from various domains including unix files and directory as well as a number of proprietry and third party tools. It also offer a good GUI interface toeasily specify wich document to include or to assembly.

However seems better designed for generate collection of documents and the granularity of inclusion was too coarse for our purpose. SoDa has some support for parts of file to be included. But these are essentially limited to single lines or fields delimited by special keys.
We did not find any ways to allow an external program to alter the generation of a document, which effectively cripples our use of Soda since we would like to work with external database application. Finally, the GUI paradigm employed by Soda, while appropriate for on-off design of documents master, did not offer the level of scripting language we needed. It would be theoretically possible to use Maker Interchange Format as a scripting language to bypass the GUI, but it would be extremely tedious and error-prone, as MIF is not designed for this.

Grif

Grif is an SGML WYSIWYG Editor developed and marketed by Grif.SA, France, based on the grif prototype developed by Inria and CNRS. Two instances of Grif restricted to HTML are freely available on the Web: Symposia [Symp95] and Thot [Thot96], made available by the W3C Consorcium.

Grif is based upon an object-oriented architecture in which each element of a DTD is considered as an objects. These objects can react to stimuli (commands) received from user via the editor. Grif also offers an API (gate) which allows to build new documents or update existing documents from applications. So it possible to define scripts to be executed when a part of a document is activated (eg. created, modify, deleted). If the action is attached to the read action, the action will be executed at this time, ie. when the document is loaded. The script can contains calls to the editor API to insert data (eg. read from a database) and new elements into the document and to alter any part of the documents.

So Grif (or any SGML editor with a rich API) seems a good candidate for being extended towards Virtual Documents. However Grif is more designed for attaching scripts to element types (ie. at the DTD level) rather than documents parts (eg. instances of the DTD). Also the standard approach requires for the code to be compiled and linkedited together with the Grif editor, which is not flexible enough for our purpose. An alternative will be to put scripts into generic annotations and to interpret them at loading time. Then we are facing back the issue of defining a general language for virtual documents as an SGML extension, for storing the document.

5.3 Related work

MacWeb[6][Nan91] synthesises documents by dynamically combining fragment of information which are organised in an object-oriented knowledge base. The links to scripts which trigger generation of the document are placed in the WebOFTypes. The synthesis of the document is done according to a task model. MAcWeb uses a script language (WebTalk) for describing the logical structure and the semantic contain of the Virtual Document. The interpretation of the script generates the Virtual Document.

The first difference with our approach is that MacWeb uses an OO knowledge base (semantic representation of the content) while we uses more standard representations of information as SGML Document databases or general databases. More importantly MacWeb generates the layout of the document from methods attached to the Objects types, making it impossible for various external applications to redesign them, as pointed out in section 3.3.

[6]The HyperWeb prototype from LRIMM, France, should not be confused with the Macintosh Web Browser of same name.

5.4 Resume of authoring features

The main requirements for a virtual document language are:

- The parts of the document which are static should be as easy to edit than usual documents, ie. it should not be generated by a program.
- All the queries are evaluated at the same time (when loading the document).
- Answers to queries can be reorganised within the document: For example a set of answers can be matched to a table or a list after being sorted.
- The answer to a query can be included under the form of the actual data or a link to the data, which account for a level of materialisation of the query.
- The virtual document specification contains control instructions for adapting the document to the actual retrieved data (eg. when there is no answer to the query) or to an user profile.
- The virtual document can include parameters by metavariables representing queries to be instanciated.
- Under an appropriate editor, the database should be updatable from a virtual document instance (a difficult problem).

When the document is loaded the virtual document is "evaluated" (using a functional language terminology) and then displayed. The evaluation may consist in generating an SGML/HTML document that will be displayed using an SGML/HTML browser.

6. Compiling versus interpreting

Although virtual documents are first thought as document that are generated on demand, it is possible to precompile them. This could be useful in contexts where it would be computationally expensive to generate them, and/or when the content is not timely sensitive, or for security reasons. One example is when the same virtual document will be used for designing Web pages as well as pages for a CD-ROM or a stand alone kiosk. The pages for the CD-ROM or the kiosk could be pre-compiled if the content is fixed. At the time of the production and cannot be updated. In this case virtual documents can be seen as a technique for specifying and producing a variety of related documents and make automatic and easier their creation, updating and maintenance.

At this stage we find useful to introduce two definitions:

An *instance* of a virtual document is the result of its evaluation or compilation, ie. the document that is displayed on the screen.
A *materialisation* of a virtual document would be the creation in the database of a document corresponding to this instance. The objects contained in this document are still dynamic and any modification to them will be reflected into the document, but new objects will not be added., and environment variables are kept constant.

A materialisation of a virtual document is a partial compilation in a specific environment. A compilation of a virtual documents for a variety of parameters can result in a set of materialised documents. For example, our list of conferences could be compiled into four lists of conferences according to the sorting criteria, would the time for loading be estimated too long. It also allows for further edition where some content of the materialised document can be suppressed.

7. Conclusions and further work

Virtual Documents offers the way of publishing and sharing the same information in a very flexible way and independently of the server or database on which this information is located. It solves then the problem of updating various documents that share the same information which will be updated only once. It would also allows users to reassemble information according to various tasks, new objectives, or different contexts, states or environments. PHP has proven to be a very convenient scripting language for building HTML virtual documents that include data from sql database. A subpart of t the DSSSL language could be a good candidate for defining more complex SGML virtual documents.

In the context of Resource Discovery a question arises of how to index Virtual Documents. A first answer is that the virtual document specification should provide a first level of meta data if the specification is declarative enough as opposed to procedural. Another possibility is to index a virtual document through its current materialisation, and to provide an incremental mechanism for updating the index.

Our future work includes the implementation of virtual documents for reusing fragments of existing SGML documents for authoring new documents or provide more customisable documents.

Acknowledgments

This work has been partially supported by the Co-operative Research Centres Program through the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia.

Bibliography

[ACM96] S. Abiteboul, S. Cluet and T. Milo, *Correspondence and Translation for Heterogeneous data*, private communication, INRIA, 1996.

[BGO90] E. Bier and A. Goodisman, Documents as User Interfaces, EP90, *Proceedings of the International Conference in Electronic Publishing, Document Manipulation & Typography*, R. Furuta Ed., P.249-262, Cambridge University Press, September 1990.

[Bro89] P.J. Brown, A Hypertext System for UNIX. Computing Systems, 2(1):37-53, 1989.

[CAC94] V. Christophides, S. Abiteboul, S. Cluet, M. Scholl, From Structured Document to Novel Query Facilities, SIGMOD 1994.

[CAT93] R.Cattell, The Object Database Standard: ODMG-93, Morgan Kaufmann Publishers, ISBN: 1-55860-396-4.

[Dua96] D.N. Duan, Distributed database Access in a Corporate Environment Using Java, Proceedings of the 5th World Wide World Conference, Paris (France), May 1996.

[Gru96] T. Gruber, S. Vemuri, & J. Rice, *Model-Based Virtual Document Generation*, Knowledge Systems Laboratory, KSL-96-16, May 1996.
http://WWW-KSL-svc.Stanford.EDU:5915/doc/papers/ksl-96-16/

[Mil096] Maria Milosavljevic, Adrian Tulloch and Robert Dale, Text Generation in a Dynamic Hypertext Environment, *Proceedings of the 19th Australasian Computer Science Conference*, Melbourne, Australia, January 1996,
http://www.cs.mu.oz.au/conferences96/abstracts/acsc.7c.2.HTML

[Nan91] M. Nanard and al, "Using types to Incorporate Knowledge in Hypertext", *Proc of 3rd.ACM Conferences on Hypertexts (Hypertext'91)*, ACM Press, San Antonio (Texas), pp.329-344, Dec. 1991.

[ODMG] Object Database Standard: ODMG-93, http://www.odmg.org/odmg-93.HTML

[Pao94] J.Paoli, Creating SGML objects for End-Users - Establishing SGML in an interactive world, Proceedings of SGML'94, GCA, Ed., pp. 323-333, December 1994.

[Rat94] Rational Software Corporation, *Framebuilder and Soda Documentation*, 1994.

[Qui94] V. Quint, I. Vatton, Making Structured Documents Active, *Electronic Publishing-Origination, Dissemination and Design* , vol. 7, num. 3, 1994.

Web References

[Confs] The Virtual Library: Conferences,
http://www.rpd.net/Info/conferences/OVERVIEW-en.HTML

[Gru96] Thomas Gruber: How Things works:
http://www-ksl.stanford.edu/people/gruber/virtual-documents-htw/salient.HTML

[Ler96] Rasmus Lerdorf: PHP/FI, http://www.vex.net/php/

[mSQL] Hugues Technology: http://hughes.com.au/product/msql/

[Nor94] Randy Norwood, *What is a document* ?, 24 March 1994
http://www.eden.com/~Ernorwood/document/virtual.htm

[O2demo] O2 Auto-demonstration
http://www.o2tech.fr:80/o2web/o2webgateway/o2www/auto_1?QuErY%3ATheService-%3Eo2_auto_report(%240)

[O2server] http://www.o2tech.fr:80/

[Peba-II] http://www-comp.mpce.mq.edu.au/msi/peba/

[Tra96] TransactNet, *Wit, The Web Interface Toolkit*, http://www.transactnet.com/

[W3C] Inserting objects into HTML, W3C Working draft, 22 April 1996, http://www.w3.org/pub/WWW/TR/WD-object.HTML