Reuse of Information through virtual documents

Anne-Marie Vercoustre [1]

INRIA — Rocquencourt,
BP 105, 78153
Le Chesnay Cedex, France

Anne-Marie.Vercoustre@inria.fr

Jon Dell'Oro, Brendan Hills

CSIRO Mathematical and Information Sciences
723 Swanston Street, Carlton,
Victoria, 3053, Australia

Jon.DellOro@cmis.csiro.au
Brendan.Hills@cmis.csiro.au

Abstract

This paper explores the issue of representing textual 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 are dynamically generated, and therefore always present up-to-date information when they are instantiated.. The benefit of this paradigm is that it allows information to be shared, reused, and adapted for various contexts. A Virtual document specification defines how to find and retrieve information objects from databases or from existing documents, and how to assemble it into another document. Virtual documents can be used to create HTML pages that contain information from one or several remote or local databases, to assembly parts of existing documents into a new one, or to define various views of the same information according to various needs. This paper focuses on the prototype implementation of virtual documents from the perspectives of document authoring and architecture. We propose an SGML syntax for Information Object that includes OQL-queries for retrieving fragments of existing documents, transformations on an intermediate tree representation, and output mapping to the virtual document structures.

Keywords Document Databases, Document Management, HTML, Hypermedia, Information Discovery, SGML, virtual documents.

1 Introduction

An enormous amount of government and corporate information now resides in electronic documents, and document management has replaced data management as a focus of computing research over the last two years. The World Wide Web has gained widespread popularity as a means to access this information, both for 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 to the right people.

When we approach a very large collection of dynamic documents, such as the WWW or even private corporate nets (intranets), our main issues for publishing information are:

- automatic updating of the information contained within documents.
- configurability of the documents according to a user's needs.
- sharing and reuse of information through various documents.
- automatically and flexibly providing meta-descriptions to deal with evolving required descriptions.

Publishing information on large networks is still mostly done by writing the information in 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 reflects 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 or information from several databases.

The Internet and the Web provide the basic mechanisms for Information Discovery in widely distributed heterogeneous networks, but, in spite of the increasing number of good Web search engines and sophisticated query interfaces, more advanced support is necessary for finding the right information for the right people, and presenting it in the appropriate form.

A new trend is to build applications which are able to directly extract information from Web pages

[1] Most of this author's work performed while working at CSIRO Mathematical and Information Sciences, Melbourne, Australia.

Proceedings of the Second Australian Document Computing Symposium, Melbourne Australia, April 5 1997.

and to reuse it to provide the user with new information more suitable for their specific need or task. For example, TransactNet has developed a tool called Wit [24] that, when given any Web page, can build and compile Java classes corresponding to the "objects" in the page. Each time this page or any page of the same model is given to Wit, Wit will construct the Java objects corresponding to the page. The objects can then be reused from other Java applications. Although WIT uses a limited and low level model of pages — built from analysing HTML tags and not supporting lists or recursive structures — it can be very useful for reusing specific Web information, for example by regularly issuing the same query to a server to check if there is something new, and only then notify the user.

Araneaus et al [1] also reuse a set of regular Web pages by building a page oriented data model to describe the schema of a structure in the spirit of database schema. It is possible to define a relational view on the server that combines path queries for extracting information according to the model, and follows links to other pages to get other information. This view can in turn be transformed into new Web pages with a different structure.

These two examples are based on the idea of reusing information from regular Web pages, i.e. pages that have been produced automatically from a database, or generated from a script. In any case the model of the pages has to be "guessed" from the HTML output. It must also be general enough to apply to a variety of Web servers, and must be recreated each time the server changes its output format, or new HTML features appear.

Our approach is to reuse information from SGML documents to take advantage of the rich structure model provided by the DTD (Document Type Definition), which guarantees that the document will conform to the model.

This paper examines the issue of reusing information as virtual documents that include data and fragments of documents from remote sources, especially from conventional 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 can be instantiated on the client side. It is an extension to the current practice of dynamically generated documents, an example of which is a CGI script for the WWW.

This paper also gives the requirements of a design language for building virtual documents. A virtual document description will include SGML-based specifications of the logical structures with OQL-like queries for retrieving content parts from a database (See Cattell [6] for a description of OQL). The virtual document description will therefore provide a constructive specification of the document.

2 Reuse of information objects

Levy [9] has identified reuse as being central to the creation and use of documents. To a great extent, the work of producing new documents involves reusing pieces of previously existing documents — where reuse involves finding relevant material, modifying it as needed, and assembling the pieces together.

The objective of our current project is to build a system for reusing SGML documents and other information sources in order to:

- create new documents from existing ones and from various sources of information
- provide customised and task-oriented views upon existing documents. (As described by Beir and Goodisman [3])
- extract information objects from SGML documents and make them accessible to distributed external applications

This approach regards a "document" as a set of information objects that can be retrieved and combined in different ways and reused in various contexts, as described by Skar [14] in his discussion of processing SGML documents as collections of small parts, rather than as monoliths.

This paper focuses on the first objective of the list above, i.e., the creation of documents in a flexible way from distributed sources of primary information.

2.1 Background

For many web publishers today, information is published in two common ways: either stored as static HTML pages, or as web pages that are dynamically produced by running scripts or programs on a web server. In the first case, the information is the HTML document itself, so that it is difficult to update, share, process, and reuse.

Hypertext and the WWW have long been regarded as good ways of sharing and reusing information, since hyperlinks 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 synopsis of several pieces of a document. Some earlier hypertext systems, such as Guide [4] offered an included mechanism for expanding links within the current page, but this required an accurate taxonomy of the set of hypertext documents, which obviously the World Wide Web cannot support.

Moreover, Web pages are rarely short and concise, and they seldom include useful structural information. For these reasons, Web pages do not offer the appropriate granularity for inclusion into other documents, as they may contain information content on a range of topics, intermixed with presentation elements that would be undesirable in another context.

HTML was initially designed for describing the organisation of the information and its semantic content, but in current day use, it is used as a presentation-oriented language. As such, HTML is inappropriate for identifying and accessing specific parts of information for a particular purpose.

It appears that information management approaches have been moving from a very rigid database approach, dealing with very structured data, to a world of flat text which is supposed to represent all kinds of information. This situation is patently unworkable. Using more structured document formats (e.g. SGML) can benefit a lot from database support, especially 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 millions 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 pages such as these, displaying a list of answers in the form of links to the actual documents and some information about the content (Title, abstract or extract of sentences etc.). Although such documents have a persistent name[2], and the content is virtual (since it can change over time), it is questionable whether a document like this can be called a virtual document or not.

Virtual documents have been defined by Gruber [8] as hypermedia documents that are *generated on demand*, in response to user input. This general definition seems at first to apply to the previous example, until we consider whether those pages are actually "documents" or whether they just look like documents. The question "what is a document?" [20] is a difficult one, but a first definition could be that a document has two main functions: to communicate information, and to assemble and organise this information to make it clear and usable to the reader.

Those pages generated by search engines only assemble information which may or may not be relevant to the user, may belong to quite different domains, or may appear in several places. 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.

Rather than the previous definition, we would define virtual documents to be hypermedia documents as dynamic documents whose structure and content can be easily adapted to the specific user. They can therefore be regarded as a flexible means 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 domains 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 [16] that displays conferences events sorted by months, call for paper, date of events, or country.
- Interfaces to databases for which a WWW gateway has been developed. They offer two different functions: database querying and updating through form-based interfaces, or database browsing by generating pages with embedded links. An example of the latter is the O2 Auto demonstration [21].
- Peba-II [23]is a natural language generation system which dynamically produces a hypertext description or comparison of animals based on a knowledge base of animals and a discourse and text planning component.
- DME [8]is a 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 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 or 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 (a unique URL) that allows a user 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 application. The end user or another application accessing the same domain database cannot reshape it

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

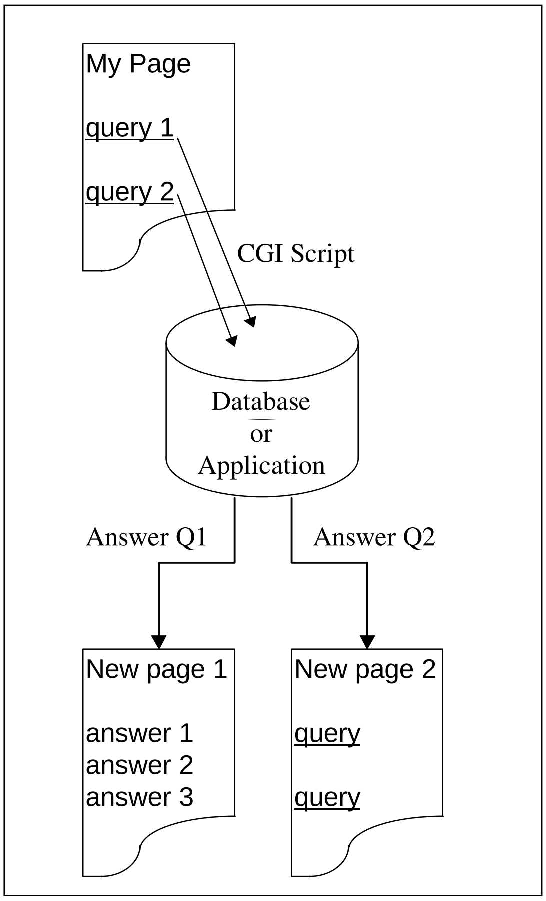

Figure 1: Server Driven Architecture

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

Typical applications which generate dynamic documents, such as those given above, issue calls to a Common Gateway Interface (CGI) program with specific parameters from an HTML document (either using a form or 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. This model is shown in Figure 1.

In Figure 1, the virtual documents *New Page 1* and *New Page 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 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. We want to take the opposite approach, i.e. a "document driven" approach, in which a virtual document combines various information chunks instantiated by queries to external applications or a databases. This approach is shown in Figure 2.

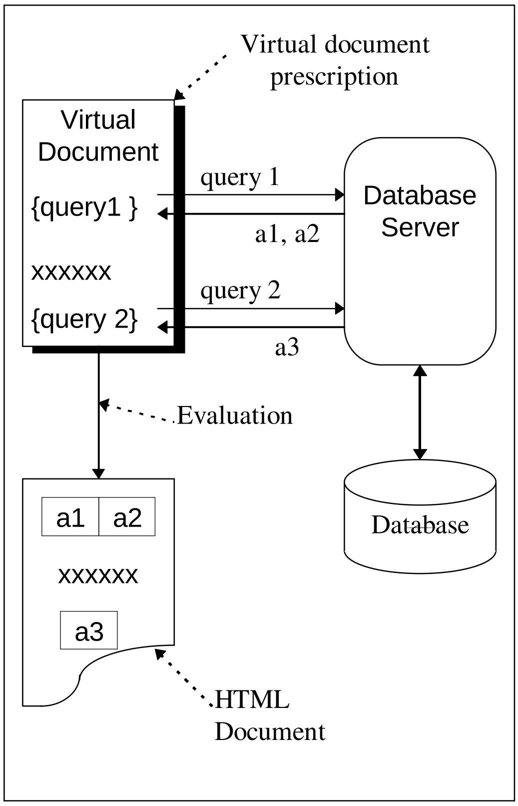

Figure 2: Document Driven Architecture

In Figure 2, the virtual document "prescription" gives any static parts of the virtual document, along with instructions about how to get the other information objects which make up the virtual document. All of the queries within the virtual document prescription are instantiated at the same time and their results are included in the document according to the instructions in the virtual document prescription. The organisation of the information within the virtual documents can be then driven by the usage of the information, rather than by the server's internal model of this information. It means also that information can be retrieved from various databases and documents, and reconfigured independent of the database developers and document authors.

3.4 Virtual documents on the Web

The model described in Figure 2 requires as a minimum a mechanism for including new elements into a document. Surprisingly, HTML does not offer this capability. More specific techniques for including pieces of information within a static HTML page include:

- an HTML document that contains the `<IMG>` tag with an URL for selecting the image to be included in the document. The URL can specify a file name or a request to a database or a CGI script that return information (e.g. the value of a counter). The WWW Consortium (W3C) is currently working on an HTML extension for multimedia objects insertion within a document through a new tag called `<OBJECT>` as a generalisation of the `<IMG>` tag [25].
- 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 to HTML that allows for displaying various sources of information in the same browsed page. Frames do not provide inclusion within the same HTML document; instead, they facilitate inclusion of several HTML pages within the same browser page in a bounded-box layout scheme.
- Java applets for inserting executable code.

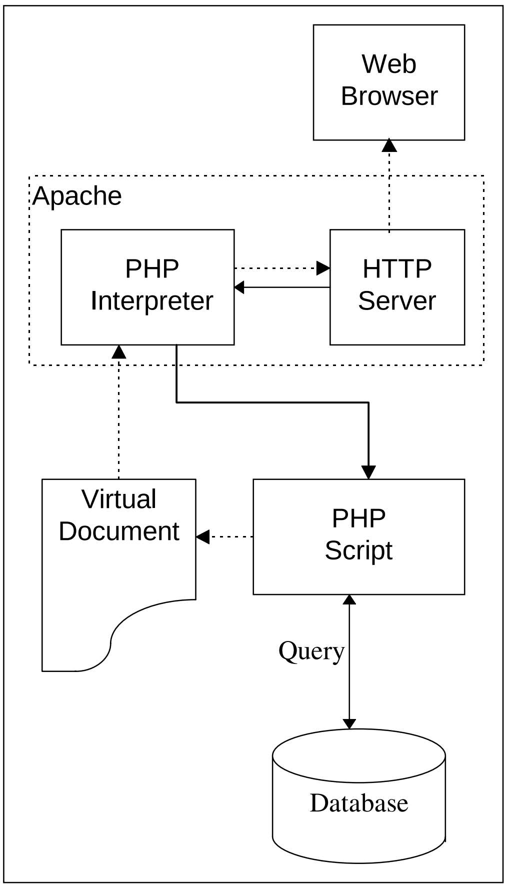

Figure 3: PHP Architecture

- advanced interfaces to databases — for example PHP as shown in Figure 3.

3.5 Virtual documents with SQL

We have worked with several examples of virtual documents to gain experience in various techniques for implementing them, both at the authoring level and at the architecture level. We will report here on a conference list with its content built from a database is a list of conferences about Hypertext, the Web, Electronic Documents, Advanced databases, Knowledge bases, etc. — all subjects of interest for our research group. The conference lists aims 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.[3]

The data for this conference list is stored in an SQL database called mSQL. This is a UNIX-based database that implements a useful subset of ANSI SQL. We access the database using a web based interpreted language called PHP/FI [18]. 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 (as shown in Figure 3), thereby running in the same process space and performing operations very quickly.

The PHP system works by embedding SQL queries and control instructions in HTML comments, which is similar to other systems such as w3-msql [19]. Our system contains only two HTML documents (actually HTML files with embedded scripting, to be processed by the PHP interpreter). The first document defines the various conference lists while the second one is a form for adding or modifying a conference entry. The first document is very complex since it controls:

- content of the document by issuing the appropriate queries according to the selected sorting criteria.
- organisation of the document by creating the appropriate grouping of information.
- 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 remind the user to be careful in editing the database.

---

[3] .The URL for the conference list is:
`http://www.mel.dit.csiro.au/~vercous/Conference-list.html`

A very small part of the Conference list virtual page is shown in Figure 4.

```
<HTML>
 <H1>Conference list for
Text Information Management</H1>
 sorted by:
<! -- open the database -->
  <?
    putenv("MSQL_TCP_PORT=1112");
    msql_connect("localhost");
    $database="confs";
  >
  <?
    $result= msql($database,
      “select Title,Location,URL
      from TIM where b_year >= 1996
      ORDER BY year,b_month,b_day)”;
      $num=msql_numrows($result);
      $i=0;
   >
<hr>
  <? while ($i<$num);
<b>
  <?
    echo msql_result
      ($result, $i, "Title");
  >
</b>
<br> <a href=
  <?
    echo msql_result
     ($result, $i, "URL");
  >
 >
  <?
    echo msql_result
     ($result, $i, "URL");
  >
 </a>
<hr>
  <?
    $i++;
    endwhile;
  >
</BODY>
</HTML>
```

Figure 4: Conference List Example

It is perhaps not surprising that the conference list which started as a very simple virtual document actually looks like a relatively complex program. To some extent, this is no better than a CGI-script.

From the architectural point of view, our virtual documents are actually scripts running on the PHP interpreter and accessing data from a database. The PHP interpreter handles a straight connection with the database (Figure 3).

This is a two-tiered architecture similar to the one described by Duan [7] for database applications on the Web using Java. Unlike JavaScript, PHP/FI is not browser dependent as JavaScript is a client-side HTML-embedded language, whilst 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 people. From the design point of view, this feature is very important, because the information reuse becomes feasible. Anyone using a PHP server could design easily their own conference list from our conference database (providing we give reading access).

From the authoring point of view, using PHP was at first satisfactory when the document were ordinary HTML documents with queries embedded into comments. It seemed so much more convenient than managing HTML and CGI files separately. Also, this enforces the document-oriented design of the virtual document, which was exactly what we were looking for. Perhaps unfortunately, we soon began to appreciate the utility and convenience of the control capabilities of the embedded programming language so our virtual document ended up looking like a program with some print statements for writing HTML tags: rather more like a CGI-script than an HTML document. Although we found this undesirable, it was perhaps inevitable because the scripting language is not declarative, but procedural.

Despite the breakdown of “declarativeness”, the scripting approach still has some advantages:

- It is very easy to construct simple documents and the result still looks like HTML.
- Complex operations can benefit by putting everything in the same file, with many control variables, to produce many different pages. The resulting pages share whatever it is possible to share and there is only one page to maintain.
- Integration with various databases is tackled by the PHP server, and you only need to know the query language syntax and your database schema.

4 Virtual documents from SGML databases

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.

As an example of this sort of documentation system, consider writing manuals for personal computers. Each model of personal computer contains a number of *components* such as hard drives, sound cards and monitors, and each component has a number of *parts* such as power supply, fan or SCSI connector. The aim of the documentation effort is to produce a complete manual for each model of computer, and for those manuals to be automatically updated when the details of a lower level part change. This would be accomplished by having a query embedded in our personal computer manual which returns the required sections of the manual for each particular component. The component manuals would then in turn have queries which return the appropriate manual pages for each of their parts.

4.1 Document granularity and storage

The storage of SGML documents in databases has been the object of much research and development to improve various aspects of document management. In particular, various query languages and systems have been proposed to combine searching for documents base on their content as well as using queries based on document structure.

The way that documents are stored in the database has a large impact on the granularity of fragments of documents that can be retrieved. The granularity for searching can be different of the granularity for retrieval. For example SIM, the "Structured Information Manager", developed by RMIT, Australia [17], stores the full SGML text while indexing documents according to part of the structure as defined by the DTD.

Christophides et al [5] store SGML documents into an object database, namely the O2 object database system [22], using a database schema that maps onto the DTD. This allows for very powerful queries on the structure of the documents and the retrieval of any fragment of document, with some extra cost in storage and current performance as search using textual predicates is not integrated in the query optimiser.

Therefore there are two levels of queries:

- queries to get a set of objects from the database by issuing a query to the database or by giving its explicit address (a URL, or a query to retrieve a specific object in SIM, O2, or the Web)
- a query within an object using the "object model": for example getting an attribute of an object, or a section of an SGML document.

To perform the latter sort of query on heterogeneous objects we need some sort of intermediary format (e.g. a generic tree). The construction of this tree will be part of the interface to the database. For example for SGML documents stored as SGML files, the tree will be built by using the SGML parser corresponding output to the DTD. It would even be possible not to build the tree corresponding to the full document, but only the parts corresponding to the queried objects, using an approach described in Abiteboul et al. [2].

4.2 Requirements for authoring virtual documents

The main requirements for a virtual document language are:

- Parts of the document which are static should be as easy to edit as usual documents. The document 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 in the form of the actual data or a link to the data, which accounts for a level of materialisation of the query.
- Virtual documents can be defined recursively, i.e. virtual document can contain queries or links to other virtual documents.
- The virtual document specification should contain control instructions for adapting the document to the actual retrieved data (e.g. 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 instantiated.
- The database should be updatable from a virtual document instance by using an appropriate editor (though this is a difficult problem).

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

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

4.3 Specification language

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

- query language for selecting and controlling the content elements.
- mapping language for transforming the previous elements into document elements.
- 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.
- presentation language.
  (see Figure 5)

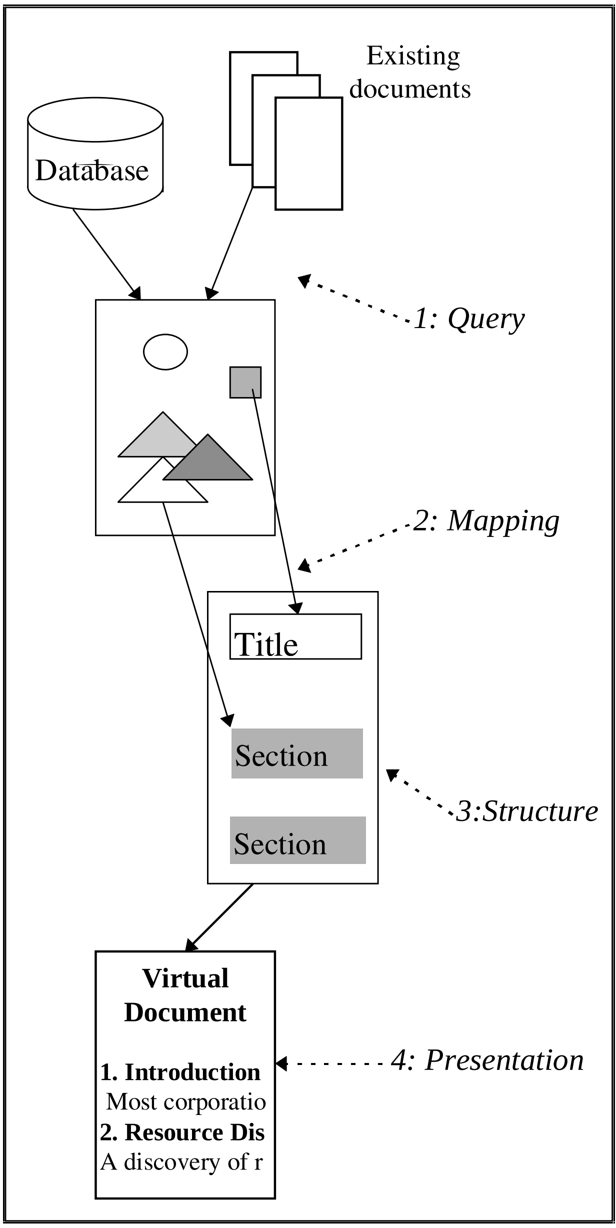

Figure 5: Document construction

The language interpretation process can be described as following:

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 specifies 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. This mapping could be defined through a middleware data model using labeled trees as proposed by Abiteboul et al. [2].
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 variables) and the actual content. Also, the construction of the logical structure of the document defines the static part of the document. The major difference of this to 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 by not redefining all the intermediary structures or common elements.
4. The presentation description specifies how to format and display the document. It also includes control instruction. If the virtual document is a 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.

The Information objects will be referenced from a virtual document prescription using the "InformationObject" tag which is designed to work in exactly the same way as any other SGML tag. An example of the InformationObject tag is shown in Figure 6.

Note that in Figure 6, the content of the `<Output>` element is an SGML structure which conforms to the virtual document's DTD at the point of the virtual document prescription which is occu-

```
<Doc>
<p>blah blah blah blah.
<InformationObject>
  <Query Method="SQL",
    Database= "Parts">
      select blabla
  </Query>
  <Transformation>
    <Process>
       (processing instructions…)
    </Process>
    <Output>
      <NumberedList>
        <ListItem>
          ${Var1}
        </ListItem>*
      </NumberedList>
    </Output>
  </Transformation>
</InformationObject>
<p>blah blah blah.
</doc>
```

Figure 6: `<InformationObject>` Tag Example

pied by the `<InformationObject>` element. This structure contains variable names at the points where the transformation process will insert data from the queried information object. This approach has been used so that the virtual document system can easily provide a "dummy" information object which means that the virtual document can be validated as correct SGML. To do this, the system simply has to remove the InformationObject tag to the Output tag, and then the Output and InformationObject close tags. This will leave the contents of the Output tag which is valid SGML.

We think that the International Standard for Document Semantic and Specification Language, DSSSL [15], could be a good candidate for defining the transformation part of virtual documents. Indeed DSSSL defines the syntax, semantic, and processing model of two languages for the specification of SGML documents processing, described here:

1. The transformation language for transforming SGML documents marked-up in accordance with one or more DTD's 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 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).

4.4 Related work

MacWeb[4][11] synthesises documents by dynamically combining fragments 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 content of the virtual document. The interpretation of the script generates the virtual document.

The main difference with our approach is that MacWeb uses an object oriented knowledge base (semantic representation of the content) while we use more standard representations of information like SGML document databases or general databases. A more important difference is that 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.

[4]This Hypertext prototype should not be confused with the Macintosh Web Browser of same name.

5 Conclusions and further work

Virtual documents offer an effective way of publishing and flexibly sharing information, independent of the server or database on which the information is located. It solves the problem of updating various documents that share the same information which will be updated only once. It 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 an SQL database. A subpart of the DSSSL language could be a good candidate for defining more complex SGML virtual documents.

6 Acknowledgements

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

References

[1] P. Atzeni, G. Mecca, P. Merialdo, E. Tabet. Structure in the Web. Submitted for Publication, 1997: `http://poincare.inf.uniroma3.it:8080`

[2] S. Abiteboul, S. Cluet and T. Milo. Correspondence and Translation for Heterogeneous data. In *Proceedings of the International Conference on Database Theory, Greece , 1997.*

[3] 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.

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

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

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

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

[8] T. Gruber, S. Vemuri, and 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/

[9] David M. Levy. Document reuse and Document systems. *Electronic Publishing*, Vol.6(4), pages 339-348, December 1993.

[10] 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`

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

[14] David Skar. Graduating from File-based to Info-based Document Construction. In SGML Asia pacific Conference, Sydney, September 1996.

Web References

[15] DSSSL: `http://occam.sjf.novell.com:8080/dsssl/dsssl96`

[16] The Virtual Library. *Conferences*: `http://www.rpd.net/Info/conferences/OVERVIEW-en.HTML`

[17] RMIT, *The Structured Information Manager:* `http://www.kbs.RMIT.edu.au/sim/sim.html`

[18] Rasmus Lerdorf: *PHP/FI*: `http://www.vex.net/php/`

[19] Hughes Technology, *w3-msql*: `http://www.hughes.com.au/product/w3-msql/`

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

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

[22] O2 Server: `http://www.o2tech.fr:80/`

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

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

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