EP98 – Saint-Malo, 1–4 April 1998 (provsionnal paper : October 28, 1997)

Reuse of linked Documents through Virtual Document Prescriptions

Anne-Marie VERCOUSTRE* and François PARADIS†

** INRIA Rocquencourt*
BP 105, 78153 Le Chesnay Cedex, France
`Anne-Marie.Vercoustre@inria.fr`
† CSIRO Mathematical and Information Sciences
723 Swanston St.
Carlton, VIC 3053, Australia
`Francois.Paradis@cmis.csiro.au`

Reuse of linked Documents through Virtual Document Prescriptions

Abstract. *As the WWW becomes a major source of information, a lot of interest has arisen, not only for searching for information, but for reusing this information in new pages, or directly within applications. Unfortunately HTML tags do not provide a significant level of structure for identifying and extracting information, since they are mostly used for presentation issues. Moreover the simple link mechanism of the Web does not support the controlled traversal of links to related pages. Particularly promising is the proposal for a new standard, XML, which could bring the power of SGML to the Web while keeping the simplicity of HTML. In this paper we present a system and a language that allow reusing information from various sources, including databases and SGML-like documents, by combining it dynamically to produce a virtual document. The language uses a treelike structure for the representation of the information objects as well as link objects. The paper focuses on the selection and the traversal of XML links to extract information from linked pages. The strength of our approach is to be an SGML-compliant solution, which makes it ready to take full advantage of XML for reusing information from the Web as soon as it is widely used.*

Keywords: : Virtual Documents, SGML, XML, Reuse, Query Language, links, OO database

1. Introduction

Reuse of information is increasingly important in a global hypermedia like the Web [14, 20]: as it becomes a major source of information, there is an increasing need for accessing that information and "building" new information from it. So far the Web community has focused on providing better browsing tools and search engines. This process of finding information is however, only the first step in an integrated solution for reuse of information. A more complete scenario for reuse of Web pages would be: i) find the pages containing the information; ii) extract the relevant information from the pages; iii) follow the links to find more information or to put this information in context; iv) use that information to build new pages, or reuse it in an application.

The extraction of information is ill-supported by HTML, because the information is semi-structured, as opposed to SGML, which allows a finer grain and more rigorous structuring of information. HTML also lacks many desirable features for following the links; it only provides simple unidirectional links, with no semantics on the links themselves. Some approaches have been proposed to make up for HTML deficiencies by adding some "SGML" intelligence to the Web [21]. Particularly promising is the proposal for a new standard, XML [9], which could bring the power of SGML to the

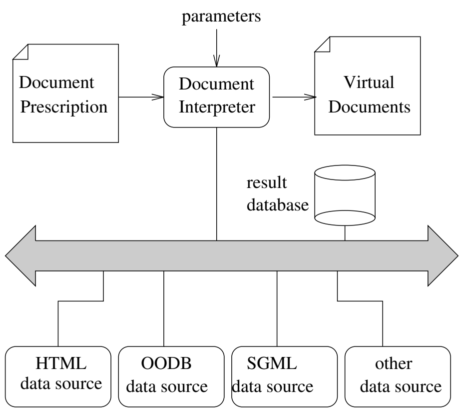

FIGURE 1: The RIO architecture

Web while keeping the simplicity of HTML. XML will facilitate the extraction of fragments from pages, and provide a comprehensive scheme for linking.

Virtual documents have been used mainly for building dynamic Web pages, with little concern for reusing existing information. As a tool for the reuse of Web pages, they provide mechanisms to integrate heterogeneous sources of information (HTML, XML, databases, etc.), and ease the task of maintaining Web sites by ensuring the information is always consistent and up-to-date.

We presents a language for virtual documents as a means to reuse Web pages, and the more specific application of reusing a set of linked XML documents. In a previous paper [23] we described two of the reuse step cited earlier: how to extract fragments of heterogeneous information and integrate them in a virtual document, which is achieved using a generic tree model. This paper focuses on how to allow and control the traversal of XML/HTML links to retrieve information that resides in linked pages. To that end, we extend the language to provide more concise and powerful syntax for link traversal.

We first introduce our language for writing virtual documents and the generic tree model for retrieving and combining information. We then present XML with a focus on the definition of links. We show through some examples how to represent, select and follow XML links using the general syntax of our language. We finally propose a new syntax for selecting and following links and illustrate it through the same examples.

```
<HTML><BODY>
< BASE HREF= "http://www.inria.fr/" >
<?pick body.table[#FIRST+1] from url("http://www.inria.fr/")>
<?define $staff as sql(select * from STAFF where portfolio="TIM")>
<H1>TIM group members</H1>
<UL><?map $i in $staff><LI><?$i.name></LI> </UL>
</BODY></HTML>
```

FIGURE 2: A simple document prescription

2. The RIO approach to virtual documents

The RIO (Reuse of Information Object) project aims to develop techniques which can support information reuse in various contexts [22]. The focus of the project is currently on the reuse of structured information from heterogeneous sources, including OO or relational databases, SGML and HTML documents, and possibly any kind of semi-structured data [3]. The overall architecture of the system is shown in figure 1.

Our approach is to introduce a *document prescription* of the virtual document to be generated. A document prescription consists of:

— Static data, or the structure and the text that do not change in the document, just like in a normal document.

— Queries, or the commands needed to generate the dynamic part of the document.

— Transformation instructions, to convert the reused information objects into new document objects.

The document prescription is written as an SGML document or as one of its derivatives such as HTML or XML. Static data is expressed using normal SGML constructs. Queries and transformation instructions are expressed as SGML *Processing Instructions*. The interpreter of the document prescription executes the following tasks:

— Sends the *native* queries to the database server which connects to the specific database, sends then queries and gets the results back.

— Assembles the results into a tree representation that provides a generic model and data structure for integrating heterogeneous data sources.

— Selects, joins and transforms objects in the tree model.

— Maps the selected objects into the document structures of the target virtual document.

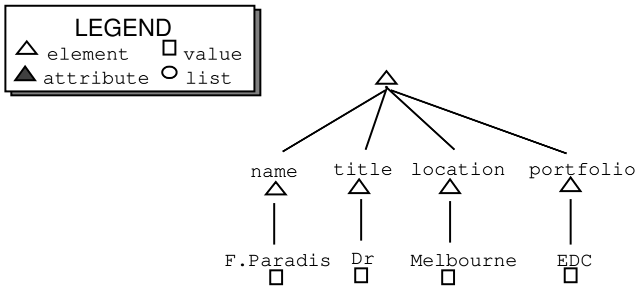

FIGURE 3: Tree representation of a person from the STAFF database

Figure 2. shows a simple example of a document prescription to generate an HTML document containing a list of staff members and some regularly updated news. The list of news is fetched from another HTML page[1], while the list of STAFF is retrieved from an SQL database of persons with names, title, location, portfolio.

The first instruction sends an http query to retrieve the HTML page and *pick*s up the second table which contains the news. The result is mapped into HTML using the function `toSGML` which unparse a subtree using its label as tag name. The second instruction defines a variable to store the results of an SQL query that retrieves all employees in the TIM group. Finally the `map` iterates over the list of employees and produces an `<LI>` element for every employee, with the employee name as content.The complete description of the language with more advanced examples can be found in [23].

The role of the *document interpreter* is to gather and combine information from the data sources, as instructed in the document prescription, and to map it according to the document DTD.[2] The integration of heterogeneous information is done using a tree-like model which is very similar to those presented in [4] or [3]. As in these approaches we are not using a model that encompasses all the source models. Instead we use a generic and minimalist approach. The data structure we use consists of ordered labeled trees. Each tree node can have one of the following types (akin to an SGML terminology): `ELEM`, `VALUE`, `ATTR` or `LIST`.

Figure 3 gives an example of the tree structure for representing an employee. More generally SQL tables are represented with an unlabeled node for each row, which has the column as children[3]. For SGML-like documents, elements are represented by

1 As this page is generated internally, copyright is not an issue here.

2 The interpreter itself does not validate the document against the DTD: that can be achieved by coupling an editor with the interpreter.

3 Of course this mapping is not unique; for an alternative see [4]

`element` nodes with a label corresponding to their tag name. SGML attributes are represented by an attribute node under the element node.

In the example in figure 2, expressions like `body.table[#FIRST+1]` and `$i.name` are *path expressions*; they perform selections on the results in an OQL-like manner (our syntax is inspired by the language discribed in [10], now named POQL). A path expression can be seen as the traversal of a tree: an expression `.L` (*dot selection*) finds the children with label `L` one level down the tree, an expression `..L` (*dot-dot selection*) finds the children with label `L` at any level down the tree.

It must be noted here that objects can form cyclic graphs, so we might need to represent links between nodes using an object identifier. However since our current focus is on SGML documents rather than general object-oriented databases, we only need to consider links between documents or between parts of document. In SGML these links are purely syntactic, so we can keep them in the tree as they are, ie. by their SGML tags.

Before showing in section 4. how to use the language to follow links, we introduce XML and more specifically XML links.

3. XML Overview

XML (eXtended Markup Language) is being developed under the auspices of the World Wide Web Consortium (W3C) to enable a subset of SGML documents to be served, received, and processed on the Web. XML offers an interesting approach that takes advantage of the powerful content-based markup and scalability of SGML while, at the same time, enabling documents to be published as easily as is currently possible with HTML. It also extends on the simple HTML link mechanism.

The main differences between XML and SGML are:

— XML is a very simplified version of SGML to make SGML tools easier to implement. As an example tag omission is not permitted: the start-tag and the corresponding end-tag must both be present.

— XML makes the difference between a *valid* document, which is a document conform to its DTD, and a *well-formed* document which only requires tags to be properly embedded. This is extremely important since it allows to process a document without its DTD, and to transmit and exchange fragments of documents without parsing the full document.

— XML offers a set of constructs for describing links between objects: in addition to the target location, they include information on their role, behavior, etc. Links are typed and may have multiple targets.

We now give more details on the definition of XML links [9] and two examples that we will use in the next section[4].

XML links can encompass both the simple unidirectional hyperlinks of today's HTML, as well as more sophisticated multi-ended and typed links.

Basically there are two sorts of link elements that can be inserted in an XML document: simple links and extended links. Linking elements are recognized based on the use of a designated attribute named `XML-LINK`. For example a simple link type named `refer` can be defined as follows using predefined attributes:

```
<!ELEMENT refer ANY>
  <!ATTLIST refer
            XML-LINK  CDATA                "SIMPLE"
            ROLE      CDATA                #IMPLIED
            HREF      CDATA                #REQUIRED
            TITLE     CDATA                #IMPLIED
            SHOW      (EMBED|REPLACE|NEW)  "REPLACE"
            ACTUATE   (AUTO|USER)          "USER"
            BEHAVIOR  CDATA                #IMPLIED
>
```

The XML-LINK attribute indicates that `refer` is a link element, here a `SIMPLE` link. The `HREF` attribute gives the location of the target resource. A locator can be more complex than a simple Url(see 4.3). Role and title are optional attributes that indicates the role of the link and the nature of the target resource, respectively. The last three attributes concern the traversal policy of the link; the first two use predefined policies that we will describe in section 5.3, while the last one is application-dependent.

A XML document with a `refer` link will contain the following text:[5]

Example 1:

```
<REFER XML-LINK="SIMPLE" HREF="http://www.inria.fr/">
          INRIA
</REFER>
```

We do not give an example of DTD for *extended* links. It will differ from a `SIMPLE` link by an `EXTENDED` value for the attribute `XML-LINK`, and the declaration of sub-elements of type `target`. An example of an *extended* link of type `see-also` would be:

4 Note however that some aspects of XML – especially the linking mechanism – are still under definition and may be revised in future releases.

5 As `SIMPLE` has been given as default value, the attribute `XML-LINK` could be omitted.

Example 2:

```
<see-also XML-LINK="EXTENDED"> see also
    <target XML-LINK="locator"
            ROLE= "Background"
            TITLE= "History of INRIA"
            SHOW= "REPLACE"
            HREF= "http://www.inria.fr/historic.html">
        History of INRIA </target>
    <target XML-LINK="locator"
            HREF= "http://www.inria.fr/chiffres.html">
        INRIA in numbers </target>
</see-also>
```

This link of type `see-also` has two targets, but could have more, each recognized by the value `locator` of the attribute `XML-LINK`.

An extended link can involve any number of resources, and need not be co-located with any of them. Dealing with links not co-located with the source is out of the scope of our language and should be addressed within a complete XML system. We are dealing only with links that are co-located with their source.

4. Managing XML links with RIO

Following links consists of: selecting the links you want to follow (section 4.1), connecting to the http server to retrieve the target documents (sections 4.2 and 4.3), and possibly `pick`ing up parts of the target documents. XML links also have *behavior* that we will discuss in section 5.3.

4.1. Representing and selecting links

When a document is retrieved, it is represented as a generic tree, as we have seen before. This applies to HTML/XML link elements as well. The link in example 1 would be represented as in Figure 4 [6].

The selection of links works exactly as the selection of any other element. If `$d` refers to the root of a document, `$d..refer` selects all the link elements of type `refer`. The textual content of the link element is obtained with `refer.#VALUE` and the target location, with `refer.href.#VALUE`. Hence links can be selected according to the values of their attributes or their anchor. The following expression selects all the links with an anchor containing the string "INRIA":

6 Note that the attributes with default value would have been added by the parser.

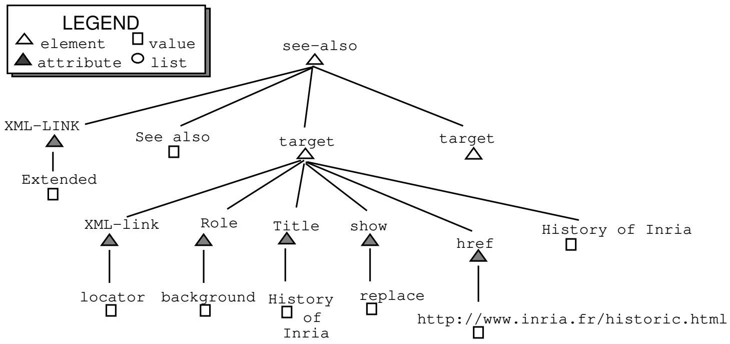

FIGURE 4: Tree representation of a link element

```
pick $link from $d..#ELEM as $link, $link.href
where $link.#VALUE contains "INRIA"
```

It is also possible to reuse the link into another link element, which is particularly useful if we want to reuse a link from XML and embed it into an HTML page (or the reverse):

```
<UL>
<?map $refer in $d..refer>
  <LI> <A> <?attributes $refer.href><? $re-
fer.#VALUE></A>
  </LI>
</UL>
```

This prescription creates a list of HTML links from all the `refer` links in the document. The `attributes` instruction adds the `href` attribute from the `refer` link (picked up with the expression `$refer.href`) to the `A` element. The value of the anchor $refer.#VALUE is reused as it is, but could be changed by any static or dynamic string, for example the name of the Url.

4.2. Following links

The aim of link traversal is to select information from other related pages. To achieve that in our language, we can use the selections shown in the previous section along with the predicate `url` to retrieve the targets. For example, the following query could be used to follow `refer` links in a document, and retrieve the target's title and URL:

Query 1:

```
pick list($d..title,$url)
from url("http://somewhere.com") as $base,
    $base..refer.href as $url, url($url) as $d
```

The path expression `$base..refer.href`[7] selects the target locations for all the `refer` links, and the expression `url($url)` requests an http connection for each of those, actually retrieving the target document [8]. The result of the `pick` is a list of tuples (title, url).

Following extended links works in a similar way. For example if we want to follow only links corresponding to a locator whose attribute `ROLE` is "background":

Query 2:

```
pick $d..title
from  url("http://somewhere.com") as $base,
      $base..see-also.target as $loc,
      $loc..href as $url, url($url) as $d
where $loc.role="background"
```

We have shown that since XML links are standard SGML elements with specific attributes, the RIO language is able to select and follow links according to their specific type, role, anchor. In the next section we study in more details the various forms of XML locators which permit to retrieve parts of documents.

4.3. XML locators

The XML syntax for locators is flexible enough to accommodate various resource locations, including resources that are not XML documents. For XML resources, XML offers a more specific syntax;

```
Locator ::= URL
Locator ::= [ URL ] Connector (XPointer | Name)
Connector ::= '#' | '|' | '?XML-XPTR='
```

Unlike HTML, XML has locators not only for retrieving full documents, but also for referring to fragments of documents, whether they are located in the current document (internal links) or a target document.

We study hereafter how the system will translate these locators into selection instructions. If the Url is provided, the resource, called the *containing resource*, is retrieved

7 a possible shortcut for `refer.href.#VALUE` in this context

8 We will see in the next section how the “location” is interpreted.

by sending an http request. If not, the containing resource is the same as the document that contains the linking element. The '|' connector is application dependent, and the `?XML-XPTR` connector is processed by the server which returns only the sub-resource. Hence we only need to consider the case where the connector is #.

`#Name` is a shortcut for `#ID(Name)`, ie. the sub-resource that has an XML ID attibute whose value matches the Name. It will be translated in RIO as :

```
pick $p..#ELEM as $p, $p.ID#ATTR."Name"
```

An Xpointer is defined by a series of location terms, absolute or relative to the previous one. For example, the locator string `CHILD(2,CHAP)(4,SEC)(3)` refers to the third child of the fourth SEC within the second CHAP within the referenced document. It will translate very naturally into the following expression: (numbering of elements starts at 0): `CHAP[1].SEC[3].[2]`.

We have shown how our language can express link traversal and retrieval of target elements contained in XML documents. The next section presents the part of the language more specific to managing HTML and XML links.

5. A specific syntax for links

The examples in the previous sections have illustrated the power of our language for selecting and following links, extracting new information objects, and finally assembling them into new documents. In this section we extend our language with specific expressions for combining path expressions and instructions for selecting and following links.

5.1. Following links

We introduce the operator "`->`" to retrieve the direct targets from a source, and the operator "`->*`", which also recursively retrieves the targets to any degree. The elementbefore the `->` can restrict the type of links to be followed. This syntax is inspired from WebSQL[16].

Using this operator, query 1 can be written as:

Query 1bis:

```
pick $d..title from url("http://somewhere.com")
     as $base, $base..refer-> as $d
```

where "`$base..refer->`" is equivalent to "`url($base..refer.href`". Table 1 gives the interpretation of some usual link expressions where `$result` refers to the results.

link expression	*interpretation*
`$base..refer->`	`url($base..refer.href) as $result`
`$base..refer->->`	`url($base..refer.href) as $d,` `url($d..refer.href) as $result`
`$base..refer->*`	Follow recursively all the refer links and return the list of target documents at any step
`$base..refer->chap`	`url($base..refer.href) as $d,` `$d..chap as $result`
`$base->`	`url($base..href) as $result`

TABLE 1: Interpretation of link expressions

The keyword `#ROOT` refers to the root of the designated tree. In `$base->#ROOT`, for instance, it refers to the target of `$base`; this expression is equivalent to `$base->`. The `#ROOT` keyword can generally be omitted, except in cases where a *dot* selection is needed after the "->" operator instead of the default *dot-dot* selection.

The danger of expressions such as `$base..refer ->*`, and how we will control it is discussed briefly in section 5.3.

The complete BNF for link expressions is:

```
link-expression ::= follow-link [ 'as' ident ]
                    follow-link 'as' ident '->' ident
follow-link ::= path-expression  link-exp
link-exp ::= { '->' [ root-expression ] }+
link-exo ::=   '->' '*' [ root-expression ]
root-expr ::=  root-expression | name
```

A root-expression is path-expression starting with `#ROOT`. The second alternative of the first rule will be used in more advanced examples in the next section.

5.2. Binding link elements

Query 1bis was deliberately not quite equivalent to query 1 since the url was missing in the result. Accessing the url will be made easier by using a more advance feature of our syntax. The query equivalent to query 1 is:

Query 1ter:

```
pick list($d..title, $anchor.href)
from url("http://somewhere.com") as $base,
    $base..refer-> as $anchor->$d
```

The expression `as $anchor->$d` binds the variable `$anchor` to the link element corresponding to the link, and the variable `$d` to the corresponding target document.

Query 2 can be reformulated as:

Query 2bis:

```
pick $d..title
  from url("http://somewhere.com") as $base,
       $base..see-also.target-> as $anchor->$d
  where $anchor.role="background"
```

5.3. Link behavior

Simple links, as well as extended links, can specify policies for link traversal, using the attributes `SHOW` and `ACTUATE`. The `SHOW` attribute can take one of the three values: `EMBED` (ie. the target resource will be embedded into the current document), `REPLACE` (the target resource will be displayed in place of the current document), or `NEW` (the target will be displayed in a new window).

With our language it is possible to create Virtual documents that change the behavior of links.

Example 3:

```
$d replace target.SHOW with attr(SHOW, "EMBED")
```

This instruction sets all the `SHOW` attributes of `target` element to the value `EMBED`.

Example 4:

```
<?define $doc as
    (pick $d replace $anchor with elem(paragraph,$embedded)
     from url("http://somewhere.com") as $d,
          $d..refer-> as $anchor->$embedded,
          $anchor.SHOW."EMBED")>
<?toSGML($doc)>
```

Example 4 is a complete prescription that takes an existing page somewhere, replaces all the links with attribute value equal to `EMBED` with the corresponding target element[9]: The fourth line of the prescription follows the link and binds the link element and the result to the variables `$anchor` and `$embedded`, as explained in the previous section. The next line selects the link elements with the attribute `SHOW` set to `EMBED`. The second process instruction simply uses the function toSGML to generate

9 We suppose that each target element can be embedded into a paragraph in the source document.

a tagged document that exactly matches the tree structure. This straight mapping involves a pre-order tree traversal, using the node labels as the tags and their value as the content.

This virtual document prescription is a little atypical. Usually a VD prescription would contain more static parts with explicit tags, and some embedded process instructions. This prescription transforms an existing page, instead of reusing parts of it.

5.4. Controlling links

Using uncontrolled regular expressions for following links can be very dangerous since it may amount to query for all the documents in the Web. [16] defines three types of hypertext links: *internal*, when the destination is within the source document, *local* if the destination and the source are on the same server, and *global* if they are located on different servers.

In RIO we are not interested in querying the whole Web, or even a large part of the Web. We know the information we are looking for *exists*, and we often know where it is, or have some idea of the structure of the pages where to find it.

Secondly we can foresee that XML will push for documents that contain typed links to support intelligent applications as pointed out in [18]. Following links according to their type will be a way of controlling the number of retrieved documents.

For these reasons we decided not to follow links that are very distant from the initial document, and the interpreter will limit recursive link traversal to target documents on the same server or domain. It will be possible though to follow an explicit finite number of global links.

6. Related research work

Other works have looked at querying structured or semi-structured information, either from a database point of view [3, 10]; from a document processing point of view, as with HyQ [2] or SgmlQL [15]; or for hypertexts [8, 7, 18].

Our selection of objects is similar to the generalized path expressions found in POQL [10] or Gram [7] ; however, their interpretation is much simplified since our paths are concrete paths into untyped tree structures that are not constrained to a database schema. This need to query semi-structured data without strict typing conventions has been recognized in Lorel [5] , which uses coercion between types to address the problem. An alternative to path expressions is tree patterns, as in SgmlQL [15]. However, as any of these languages, SgmlQL does not address the problem of conversion to a different DTD nor to deal with heterogeneous data.

This problem of DTD "conversion" has been addressed already by a number of approaches: either by general-purpose languages such as DSSSL [1], or within SGML editors for supporting cut-and- paste between incompatible elements [6]. Conversion will be important for reusing XML documents from HTML or the reverse.

Various languages for querying the WWW in a database style have been developed in W3QL [13], [16], WebSQL [19]. Their approach is to build a model for Web pages as well as for links and to query the Web as an SQL database. Araneus [19] models servers that present a regular structure with a nested tree model mapped into SQL tables. WebSQL models HTML pages as tuples [url, title, text, type, length, modif] and links as [base,href, label), and define an SQL-like language for querying pages and following links. To control the navigation they emphasize the distinction between local and remote documents. Our approach differs of these approaches in making full use of the SGML/XML tags for exploiting the internal structure of the document. To control links we mostly rely on link types.

Building Web pages with dynamic content can be done using the limited HTML language facilities (server side *include*, `<IMG>` tags, `<OBJECT>` element), with JavaScripts or JavaApplets, or with Editors like WebWriter [11]. Advanced servers, like PHP[10] offer an interface for accessing SQL databases from HTML. This works by embedding SQL queries and control instructions in HTML comments that are preprocessed on the server. Most of these approaches have the disadvantage of being script-based, which makes it difficult for non programmers to create pages, and for anyone to update them or to include them in other documents.

Language for reusing Web information have been less explored.

XML itself offers capabilities for reusing fragments of XML documents, by using `AUTO`, `EMBED` links and by supporting XML documents that are only well-formed, relaxing the constraint on the conformance to a DTD. However it does not allow for restructuring or mixing the reused information.

Araneus [19] offers a limited model for creating new Web pages after querying existing ones. The constructed pages can only include text, lists and links to other pages, and not the full-range of HTML constructs; from the definition of new pages, a new set of static HTML pages are created. Boomerang [12] uses page templates and string pattern matching based-rules for reconfiguring Web pages. Rules are used to instantiate variables which are then expanded in the templates. Boomerang does not make real use of the structure of the document and does not allow for merging information from several pages.

WebMethods[11] relies on using HTML tags for building Java objects that can be directly used from applications. Although it has been one of our initial sources of ins-

10 R. Lerdof, PHP/FI documentation,http://www.vex.net/php/doc.html.

11 WebMethods,The Web Interface Toolkit, http://www.webMethods.com/.

piration, it can't follow links and does not offer any language for building new documents nor integrate various sources of information. Dkweb [24] proposes an architecture for dynamically customizing the structure and the content of Web pages, by storing view definition and meta information objects into an OO database. These views are a simplified form of our prescriptions, while embedded methods turn Web pages into information objects. Our tree model offers a more generic representation of information objects.

MacWeb [17] synthesizes documents by dynamically combining fragments of information that are organized in an object-oriented knowledge base. It uses a script language for describing the structure and the semantic content of the virtual document, which makes it difficult to author the document specification, even for the static parts.

7. Conclusion

Reusing information contained in electronic documents is becoming a major issue, whether it is proprietary information or information available from the Internet. In this paper we have presented a language for reusing information objects from heterogeneous sources, including SGML, XML, HTML documents. Our approach is to use a middleware format to integrate the results of queries from the various sources and to map its into a new (virtual) document. We have demonstrated more specifically how to use the language for following XML links and to control the traversal of links using their types and properties. Since our solution is generic and fully SGML compatible we are ready to benefit the intelligence that XML, or any HTML extension, will bring to the Web for supporting the extraction and reuse of information.

The approach is currently being implemented using Java for the interpreter and the database server; our prototypal application is a virtual document prescription for generating activity reports that reuse information from our Intranet, an SQL database of staff and an OO database of documents. Other potential applications are: flexible and manageable generation of large documentation, or configuration of Intranet servers.

Further extensions to the language would include control instructions to make the virtual document more adaptable to the actual results of queries, and explicit instructions for building a set of related pages.

References

[1] ISO 10179. *Document Style and specification Language (DSSSL)*, 1996.

[2] ISO 10774. *Information Technology- Hypermedia/Time-based Structuring language (HyTime)*, 1992.

[3] S. Abiteboul. Querying semi-structured data. In *Proceedings of ICDT'97 (Invited talk)*, 1997.

[4] S. Abiteboul, S. Cluet, and T. Milo. Correspondence and translation for heterogeneous data. In *Proceedings of ICDT97*, 1997.

[5] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semi-structured data. *Journal of Digital Libraries*, 1(1), 1996.

[6] Akpotsui, V. Quint, and C. Roisin. Type modeling for document transformation in structured editor. *Mathematical and Computer Modelling*, 1994.

[7] B. Amann. Gram: A graph data model and query language. In *Proceedings of the Second European Conference on Hypertext, ECHT'92, Milan*, 1992.

[8] C. Beeri and Y. Kornatzky. A logical query language for hypertext system. In *Proceedings of the first European Conference on Hypertext*, pages 67–80. Canbridge University Press, 1990.

[9] T. Bray and C.M. Sperberg-McQueen. Extensible markup language (XML), W3C working draft, http://www.w3.org/pub/WWW/TR/WD-xml.html.

[10] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proceedings of the ACM SIGMOD Conference on Management of Data, Minneapolis, Minnesota*, pages 313–324, 1994.

[11] Arturo Crespo and Eric A. Bier. Webwriter: A browser-based editor for constructing web applications. In *Proceedings of WWW5 Conference, Paris, France*, May 6–10 1996.

[12] Curtis E. Dyreson and Antony M. Sloane. The boomerang white paper: a page as you like it. In *Proceedings of the WWW4 Conference*, pages 667–676, December 1995.

[13] D. Konopnicki and O. Schmueli. W3QS: A query system for the world wide web. In *Proceedings of VLDB'95*, pages 54–65, 1995.

[14] D. M. Levy. Document reuse and document systems. *Electronic Publishing*, 6(4):339–348, December 1993.

[15] Jacques Le Maitre, Elisabeth Murisasco, and M. Rolbert. From annotated corpora to databases : the SgmlQL language. In *CSLI lecture notes, collection linguistic databases*. Cambridge University Press, to appear.

[16] A. Mendelzon, G. Mihaila, and T. Milo. Querying the world wide web. In *Proceedings of PDIS'96, Miami, Floride*, 1996.

[17] Jocelyne Nanard and Marc Nanard. Using types to incorporate knowledge in hypertext. In *Proceedings of the 3rd ACM Conference on Hypertext, ACM Press, San Antonio (Texas)*, December 1991.

[18] Marc Nanard and Jocelyne Nanard. Should anchors be typed too? an experiment with macweb. In *Proceedings of the ACM Conference on Hypertext, HTX'93, Seattle*, November 1993.

[19] P. Merialdo P. Atzeni, G. Mecca. To weave the web. In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB'97)*, 1997.

[20] D. Skar. Gratuating from file-based to info-based document construction. In *Proceedings of the SGML Asia Pacific Conference, Sydney, Australia*, September 1996.

[21] C. M. Sperberg-McQueen and Robert F. Goldstein. Html to the max: A manifesto for adding SGML intelligence to the world-wide web. *Computer Networks and ISDN Systems, 28*, pages 3–11, 1995.

[22] Anne-Marie Vercoustre, Jon Dell'Oro, and Brendan Hills. Reuse of information through virtual documents. In *Second Australian Document Computing Symposium, Melbourne Australia*, pages 55–64, April 5 1997.

[23] Anne-Marie Vercoustre and François Paradis. A descriptive language for information object reuse through virtual documents. In *4th International Conference on Object-Oriented Information Systems (OOIS'97), Brisbane, Australia*, 10–12 November 1997.

[24] Jack Jingshuang Yang and Gail E. Kaiser. An architecture for integrating OODBs with WWW. In *Proceedings of the WWW5 Conference, Paris, France*, May 1996.