Hypertextual Navigation in the SgmQL Language: 
Browsing, Querying and Restructuring Web-like Networks

Emmanuel Bruno, Jacques Le Maitre, and Elisabeth Murisasco 
GECT, Equipe Systèmes d'Information MultiMédia 
Université de Toulon et du Var, BP 132, 83957 La Garde cedex France 
{bruno, lemaitre, murisasco}@univ-tln.fr

Article received on November 15, 1998; accepted on April 19, 1999

Abstract

Due to the growth of networks, the notion of document has evolved to the notion of hyperdocument. It is more and more usual to fragment documents in a set of files referencing each others. These files can be distributed over a network. The manipulation of the whole document requires the ability to browse this graph-like structure. That is why we need tools that are able to browse these structures, to query them and eventually to restructure them. SgmQL is an OQL-like functional language which manipulates SGML documents. We present in this paper an extension of this language to express a controlled navigation on Web-like networks.

Keywords:
Query language, SgmQL, Web, SGML, XML.

1 Introduction

Due to the growth of networks, the notion of document has evolved to the notion of hyperdocument. It is more and more usual to fragment documents in a set of files referencing each others. These files can be distributed over a network. The manipulation of the whole document requires the ability to browse this graph-like structure. Manual navigation remains the more usual way to browse it either from a known URL or from the output of an index server. Nevertheless, it is difficult to avoid digressions and even getting "lost in the cyberspace"; it is mandatory to read each accessed document to check its reliability. Moreover, it is difficult to take advantage of hypertext specific organizations such as lists or trees. That is why we need tools that are able to browse these structures, to query them and eventually to restructure them.

Some languages have been proposed to deal with documents distributed over a network (Konopnicki and Shmueli, 1995; Laksmankan et al., 1996; Mendelzon and Milo, 1997; Vercoustre and Paradis, 1998). Most of them focus more on the expression of the navigation between documents than on the operators to query reached documents. Our proposition is to add navigation capabilities to an existing query language for documents: SgmQL (Le Maitre et al., 1996), therefore every component of each document can be used to control the navigation. Moreover all these components can also be extracted and reused in new documents. SgmQL is one of the languages (Blake et al., 1994; Christophides et al., 1994; Yan and Annevelink, 1994) developed to manipulate SGML (Goldfarb, 1990) documents.

SgmQL is an OQL-like functional language which can be used with any kind of document from which a tree-like structure can be extracted, in particular SGML, HTML, XML (XML, 1998) or even RTF. SgmQL enables to extract parts of documents and also to construct new documents by means of tree traversing and tree transformation operators. Its distinctive feature is that it
does not use a transcription into a classical database schema, and that only a partial knowledge of the structure of the document is needed to query it.

We present in this paper an extension of the SgmIQL language. The purpose of this extension is to express path filters and to use them by means of a specific operator. This operator controls the navigation through a set of constraints on the followed path, on the links and on the content of traversed documents. Section 2 shows how a path filter is expressed and used in a SgmIQL query. Section 3 focuses on the problem of cycles. Section 4 concludes.

2 An Operator for Controlled Navigation

In a hypertext, hand-browsing is the usual way to access documents. However it is difficult to move on a graph-like structure with a sequential access, while keeping the hypertext structure in mind and not to get lost because of the huge number of documents and the existence of cycles. Moreover, it is not possible to ensure that the navigation is exhaustive and that the constraints which the user wants to apply, are continually checked.

2.1 Path Filters

In (Bruno, 1998), we proposed to separate the start of the navigation from the path to follow. Therefore, we view a path like a list of pairs (link, node).

The link component catches all the data needed to define the relation between the source and the target documents. This component is represented as an SGML element provided with the attributes base and href, which refer to the source and the target documents, label, which labels the link, rel and rev, which specify the semantics of the relation between the source document and the target document and its inverse.

The node component is built from the document which is viewed like a node of the graph. The root tag is enriched with the following attributes: url which identifies the document, mimetype, length and modif, which give its date of creation or its date of last update.

We can now define path filters as regular expressions:

- If \( F_1 \) and \( F_2 \) are path filters then \( F_1 \ | \ F_2 \) filters any path with is filtered either by \( F_1 \) or by \( F_2 \).
- If \( F \) is a path filter then \( F^* \) filters either the empty path or a path composed of \( n (n>0) \) consecutive subpaths filtered by \( F \). To impose a bound on the time \( l \) and the length \( l \) of the navigation, an \( F^* \) filter can be suffixed by \( [l, f] \). These two parameters ensure that the number of filtered paths is finite.

For instance, the expression \((\rightarrow, _) \ (\rightarrow, _)\) filters a path of length 2. Variables can be bound to any component of a path filter using the keyword as. These variables are used to define constraints on the followed links or on the content of the documents accessed during the navigation. They can also be used to extract some components of these links and documents. Note that variables in an \( F^* \) filter cannot be seen outside \( F \) (see section 2.3).

2.2 The Navigate Operator

In order to produce the set of all paths matching a given pattern from a start document, we propose a specific operator called navigate. The expression “navigate \( u F \)” starts from the url \( u \) and for each path filtered by \( F \), instantiates the variables of \( F \). Therefore, the navigate operator produces a list of environments. For instance:

navigate “http://www.univ-tln.fr”
\[ \rightarrow \text{as}$1$, \text{as}$d) \]

- filters all the paths of length 1 starting at the url http://www.univ-tln.fr,
- produces the list \( \{[l=v_1, \text{sd}=v_2], \ldots, [l=v_n, \text{sd}=v_2]\} \) if \( n \) paths are filtered and if for each filtered path \( l, \text{v}_1 \) is a link element and \( \text{v}_2 \) a node document.

This operator can be used in the from clause of a SgmIQL query. For example:

\((Q1)\)
select first TITLE within \( \text{sd} \)
from navigate “http://www.univ-tln.fr”
\[ \rightarrow \text{as}$1$, \text{as}$d) \]
where \( \text{sd}$1\rightarrow$label match “java”

returns the list of the titles of the documents reached from the Web server of the University of Toulon through a link whose label contains the word “java”.

The user can use a easy-to-read presentation: the expression \( \rightarrow \text{as}$\text{var}$ \) is the same as the expression \( \text{var} \) and the expression \( \text{as}$\text{var}$ \) is the same as the expression \( \text{var} \).

The query \((Q1)\)bis built on top of query \((Q1)\) shows how new SGML elements or documents can be constructed.

\((Q1)\)bis
\text{document body}:
let $\text{result} =$
2.3 Guards

In the above mentioned example, the constraint on the link is set in the where clause. Setting the constraints in this way delays their checking after the generation of each path. This causes two kinds of problems: optimization (it is better to check constraints during the navigation) and expression (when arbitrary length filters have to be constrained). One solution can be to add guards to filters as it is done with the Caml functional language (CAML). A guard is a condition added to a filter in order to check constraints during the navigation. In SgmQL, a guard is introduced by the clause when. For example, using a guard, query Q1 can be optimized as follows:

\[ Q_2 \]
```
select first TITLE within $d
from navigate "http://www.univ-tln.fr"
($l when $l->LABEL match "java", $d)
```

The label of each link is now checked during the navigation, and only needed documents are loaded in main memory.

Let \( F^* \) be a filter. Guards are mandatory to set constraints on \( F \). Let us consider, for example a linked list of documents (each document contains a link to the next one). The query selecting the title of each document from this list can be expressed as follows:

\[ Q_3 \]
```
select first TITLE within $d
from navigate $url_start
($l when ($l->REL match "next"), _)* (1)
(_, $d) (2)
```

Lines (1) and (2) provide access to each document within the list (the variable $d will successively contain each document). Line (1) filters each path starting at $url_start of length 0 up to \( n \) (number of documents in the list). For each filtered path, line (2) gives access to the last document in the path (null link =).

2.4 Using Index Servers

Finally, like WebSQL (Mendelzon and Milo, 1997), SgmQL enables to call an index server (Altavista for instance) to select documents about a given topic by means of a specific operator: about. For example, the following query selects the title of the documents dealing with Java beans referenced by documents about Java provided by an index server:

\[ Q_4 \]
```
select first TITLE within $d
from navigate $url about "java",
($l when ($l->label match "beans"), $d)
```

3 Dealing With Cycles

In a hypertext a lot of documents are referring to each other. Consequently it is necessary to be able to deal with cycles to avoid infinite navigation. The first solution could be to forbid traversing twice the same document. This is easy to implement but is semantically insufficient because the same document can be differently processed at different steps along the navigation.

The best solution would be to take into account the way each document is accessed: not only the link reaching the document but also the whole navigation before. This needs an in-depth study of the semantics of the links. This remains to be done. Between these two solutions we have chosen to keep track of the links reaching a document and to avoid returning to the same document by the same link.

Note that if a dangling link occurs along a navigation (the target document does not exist) this path is left and next one is tried.

4 Conclusion

Because of the increasing use of hyperdocuments over networks, it is essential to provide tools that make their localization, extraction and reuse easier.

We have presented in this paper an operator (Bruno, 1998) in order to manage fragmented data easily. This operator uses a path filter with associated constraints, to carry out a controlled navigation over a hypertext from either a start point (a valid URL) or the URL returned by an index server. Combined with standard SgmQL operators, the user can express a query which:

- specifies a navigation where conditions are set on the followed path, and on the traversed links and documents,
- extracts hypertext fragments,
• reuses these fragments with classical SgmQL operators to build new documents (or hyperdocuments).

This paper also focuses on the problem of cycles and presents a detection mechanism. Finally, a language to express constraints with guards is proposed: i) to check these constraints in arbitrary length paths; ii) to check constraints as soon as possible during the navigation. All the functionalities described in this paper are implemented and validated (SgmQL). Other languages like WebSQL (Mendelzon and Milo, 1997), Weblog (Laksmanan et al., 1996), W3QL (Konopnicki and Schmueli, 1995) or RIO (Vercoustre and Paradis, 1998) deal with the same set of problems. Our extension is close to WebSQL or RIO. However our navigate operator is part of the SgmQL language. Therefore a query can constrain any hypertext component and build a new document, without any translation in a classical database schema. The perspectives of our work are to produce new hypertext in order to offer dynamic views (which could be specific to user profiles), to add connecting capabilities with servers processing documents to add meta-data and to study the semantics of links, in particular to solve the cycles problem.

References


XML http://www.w3.org/TR/REC-xml

Yan T. W. and Annevelink J., “Integrating a Structured-Text Retrieval System with an Object-Oriented Database System”, Proceedings of the 20th Conf. on Very Large Data Bases (VLDB ’94), Santiago (Chile), 1994.

Emmanuel Bruno is a Ph.D. student in Computer Science at the University of Toulon. His research deals with information systems and databases. In particular, he studies a way to query semi-structured distributed and fragmented documents. He is a member of the SIMM (Multimedia Information Systems) team.

Jacques Le Maître is a Professor in Computer Science at the University of Toulon. He leads the SIMM (Multimedia Information Systems) research team of the GECT laboratory. His research interest include databases, structured documents and functional languages.

Elisabeth Murisasco is an Assistant Professor in Computer Science at the University of Toulon. Her current research topics are textual databases and distributed hypertext databases. She is a member of the SIMM (Multimedia Information Systems).