Webspace query formulation: an overview

Roelof van Zwol Peter M.G. Apers

University of Twente,
Centre for Telematics and Information Technology,
Department of Computer Science
P.O.box 217, 7500 AE, the Netherlands
phone: +31(53)4898027, fax. +31(53)4892927
{zwol, apers}@cs.utwente.nl

Abstract

To find information on the World-Wide Web (WWW), two approaches are generally followed. Browsing the web from a specific starting point, or web-site map, is called search by *divergence*. The second approach, search by *convergence*, is followed when using a search engine. Most search engines use a information retrieval strategy, which requires that the user supplies some keywords to find the relevant information. Due to the diversity and unstructuredness of the WWW, both approaches offer only limited query formulation techniques to find the relevant information. When focusing on smaller domains of the Internet, still large collections of documents have to be dealt with, which are presented on a single web-site or Intranet. There the content is more related and structured, which allows us to apply database techniques to the web.
The Webspace Method aims at using DB techniques to model and query such document collections. A semantical level of abstraction is obtained, by describing the content of the documents with some high-level concepts, defined in an object-oriented schema. This allows us to bring the power of query formulation as known within a database environment to the web. At the same time, we focus on the integration with Information Retrieval, which allows us to formulate complex content-based queries over a collection of web-based documents, containing various types of multimedia.
After an introduction into the Webspace Method, the focus in this article will be on the formulation of complex queries over a collection of related multimedia documents, also called a webspace. For that purpose the Webspace Search Engine is built, which combines search by both divergence and convergence to formulate the query, using a graphical representation of the webspace schema. Under the hood, the Webspace Search Engine uses the Data eXchange Language (DXL) to gather the requested information. We will explain the DXL's framework for data exchange, and discuss how it is integrated into the Webspace Search Engine. Furthermore, we will show by some examples how, with help of the Data eXchange Language (DXL), specific parts of documents can be retrieved and integrated into the result of the query, based on the concepts defined in the webspace schema. This in contrast to the average search engine, which just delivers a document's URL.

1 Introduction

Finding relevant information on the World-Wide Web (WWW) is often a frustrating task, due to its unstructured nature. Moreover, from a modeling point of view, the topics dealt with on the Internet are too diverse to capture (model) in a database schema. Besides being too diverse, the data is often irregular, probably incomplete, and changing rapidly. These properties make it infeasible to apply database techniques directly for the Internet at a large scale basis. However, when focusing on smaller portions of the WWW, database techniques can be successfully invoked for the search process. Within such domains large collections of documents can be found, containing related information. Although the conditions are better, one still has to deal with the *semi-structured* and *multimedia* character of the data involved. The Webspace

Method focuses on such domains, like Intranets and large web-sites. Based on an object-oriented schema it provides an approach for (1) modeling web-data, (2) meta-data extraction and multimedia indexing, and (3) query formulation over a document collection, i.e. a webspace.
As presented in this article, the main contribution of the Webspace Method is that it provides a new approach to query web data. This is achieved by providing powerful query formulation techniques, as known within a database environment. Secondly, it allows users to directly extract the relevant parts of one or more documents as the result of a query, instead of a document's URL.
The object-oriented schema, also called the *webspace schema*, contains concepts that identify the content of a webspace at a semantical level of abstraction. Documents are then modeled as a view on the webspace schema using XML and are likely to contain various types of multimedia. To query a webspace relevant information should be extracted from the documents. The Webspace Method extracts this conceptual and multimedia meta-data from the documents and stores it in a object server. Finally, a webspace is queried with the Webspace Search Engine, which uses the webspace schema to formulate complex queries over a webspace.
The focus in this article is on the query formulation framework used for the Webspace Search Engine. We will show how users can formulated complex queries, with help of a graphical user interface. Once the necessary information is gathered, the query needs to be constructed and evaluated. For this purpose we use the Data Exchange Language (DXL). DXL provides a framework to query heterogeneous sources. The information gathered is then integrated in a single target. In case of the Webspace Search Engine, there are two types of sources, the object server containing the meta-data and the XML documents, which describe the content of a webspace. The result of the query is again an XML document, containing the requested information. In contrast to the regular search engines the results do not necessarily yield the document's URL, but it can contain more precise information extracted from one or more documents. We will explain the Webspace Method, i.e. modeling a webspace, meta-data extraction, and in particular the query formulation part, using the Australian Open webspace[1].

State of the Art

In literature we have found two other systems that are closely related to the Webspace Method. The first is Araneus[11], which focuses on the modeling of web sites, using a relational approach. The system, the Object-Web wrapper [9], focuses more on querying web-data, based on object-classes. The object classes define a view, which is incorporated in the search process. Both systems, only cover a part of the approach followed by the Webspace Method. In case of the Araneus project, this concerns the web-data modeling part. Instead of a relation approach, we use an object-oriented approach, and prefer to store the data in XML documents, rather than a DBMS. Our object server only stores the meta-data involved. Although we prefer the storage of data in XML, we can just as easily store the data in a DBMS. In case of the Object-Web Wrapper, the modeling part in left out. There the focus is more on the query process. With use of views, defined in terms of object-classes, queries can be composed to search a domain specific data collection. But, both systems do not use a meta-data approach for searching the document collection. Furthermore, the Webspace Method aims at integrating conceptual modeling with information retrieval, which is also not the case there. Some other approaches, like XQuery [16], XML-QL [4],and other [12, 3] do not use a central schema, but try to exploit the structure of the XML documents directly. This allows them to search for patterns and structure in the XML data, but does not allow them to formulate content-based (IR) queries over the various types of multimedia which can be found on the web. In [6, 8], about XIRQL and searching text-rich XML documents with relevance ranking, the focus is on XML and information retrieval. These approaches do not use a conceptual model, and use only a partial integration of the IR model. Of course in the field of information retrieval and multimedia databases many sophisticated models are proposed. We do not aim to come with better IR techniques, but aim to combine existing IR techniques

[1]The original Australian Open web site can be found at http://www.ausopen.org/. The Australian Open webspace is generated, using the Feature Grammar Engine [17]

with conceptual modeling, using a database-oriented approach. For those interested in information retrieval and MM-DBMS, we refer to [1, 7, 14], where these matters are discussed.

Organization of Paper

In Section 2 two aspects of the Webspace Method, modeling web data and meta-data extraction, are discussed shortly to provide the necessary background for query formulation over webspaces. How queries are formulated, using the Webspace Search Engine is presented in Section 3, while Section 4 discusses the construction of the query, using DXL as a framework. Finally we will come to our conclusions in Section 5.

2 The Webspace Method

The Webspace Method for dealing with web data, i.e. modeling and querying web-based document collections, covers three aspects.

- Modeling web data, based on the principles of conceptual modeling.
- Meta-data extraction of conceptual information, and multimedia indexing.
- Querying web-based document collections, i.e. a webspace.

The first two aspects will be discussed shortly in this section, to provide the necessary background for querying a webspace. The architecture of the Webspace System, as shown in Figure 1 implements the Webspace Method. The architecture shows how the three aspects of the Webspace Method are related. A more detailed discussion of the architecture can be found in [19].

2.1 Modeling Web Data

Modeling the content of a web site, or Intranet, as proposed by the Webspace Method, provides a better organization of the content involved, which makes it easier to present and maintain the information. In [18, 19] we presented a formal model, and the Webspace Modeling Tool used by the Webspace Method. For each webspace, a (finite) set of concepts is defined, that describes the content of a webspace (a collection of documents) at a semantical/conceptual level. These concepts are then modelled in an object-oriented schema, to define the relation between concepts. To be more specific, concepts are defined in terms of classes, attributes of classes, and associations between classes. Figure 2 shows a fragment of the webspace schema, set up for the Australian Open webspace. There, the concepts *player*, *match*, and *report* are defined as class concepts. While the concepts *name*, and *country* are attributes of the class concept *player*. Finally, the class concepts *player* and *match* are associated by the concept *is_player_by*. When modeling web data, two aspects need to be taken into account. First of all, the semi-structured character of the data involved. Most of the time the information available is incomplete. But secondly, web-data is also likely to contain various types of multimedia. These multimedia objects are also described by conceptual classes and integrated into the webspace schema, such as Image, Video, and Hypertext in case of the Australian Open webspace. At the document/physical level documents are defined, which form a view on (a part of) the webspace schema. The documents are marked up with XML, and form a materialized view over the webspace schema, since they contain both schema and content. Alternatively, it is also possible to store the content of a webspace in a DBMS, as proposed by [11, 13, 5].

2.2 Meta-data Extraction and Multimedia Indexing

Before a webspace can be queried, it is necessary to fill the object server with meta-data, which is obtained from the concepts used in the document. At this point we introduce the term *web object*, which is the

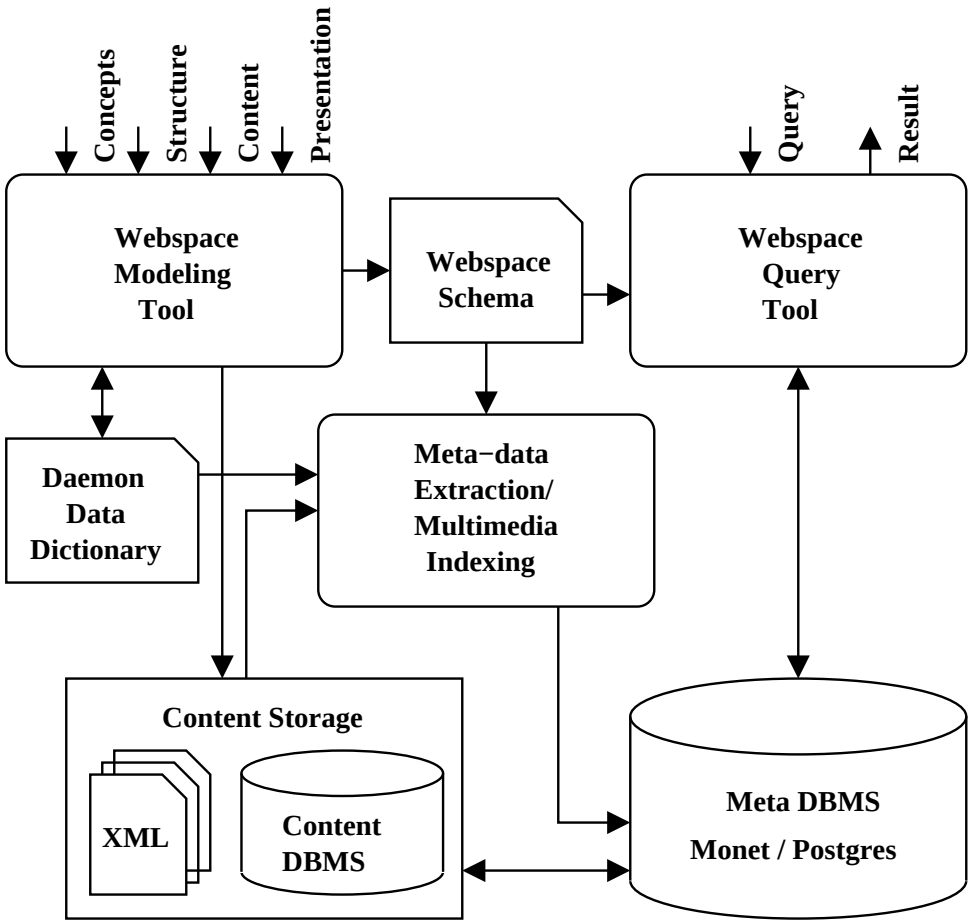

Figure 1: Webspace System Architecture.

instantiation of a conceptual class. Note that we also consider a hypertext-fragment, video, or image to be a web-object. Since each document is a materialized view on the webspace schema, web-objects can be extracted from a document by mapping the document onto the webspace schema. Next the meta-data is extracted from these objects and stored in the object server. At this point we are able to define conceptual queries over a webspace. However, to be able to formulate content-based queries over the multimedia involved, we need to build a multimedia index, as proposed in [14], and integrate this in the conceptual model. A complete description of this process is presented in [20], where the integration of multimedia retrieval into the conceptual model is discussed.

3 Formulating the Query

When focusing on query languages for web data, it is next to impossible to neglect the rapid developments around the XML standard. Query languages, like XQuery, XML-QL, and others have been introduced to explore XML's structural capabilities. With such query languages it becomes possible to deliver the requested information in the form of a newly defined (XML) structure, rather than the limited presentation forms that are currently used by the search engines. They often just return the document's URL, accompanied with some small summaries.
However, exploiting these new capabilities is not as easy as it seems. First of all, this requires knowledge of the document's structure, which is unknown for a random document on the WWW. Secondly, it requires the tools to be available for an average user to compose such a relatively complex query. The first problem

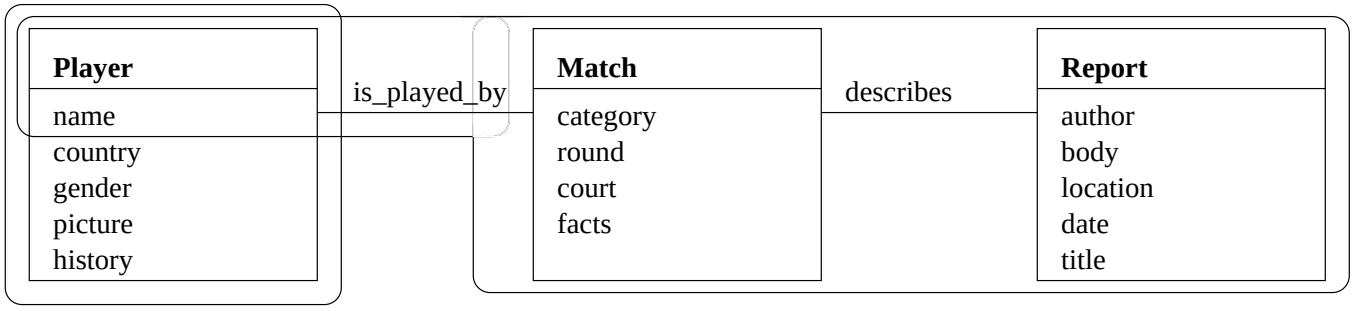

Figure 2: Fragment of the Australian Open webspace.

is completely covered when using the Webspace Method. The formulation problem of complex queries over a webspace at the **conceptual** level is covered by the graphical user interface (GUI) of the Webspace Search Engine, which is discussed in Section 3.1.
A third problem remains: the gap between the conceptual level and the physical level. Once a query is formulated at the conceptual level, it needs to be translated into a form acceptable at the physical level. In case of the Webspace Search Engine, this means that information needs to be gathered from (at least two) heterogeneous sources into a single result. To be more precise, first the meta-database needs to be queried, to find out where the conceptual information is stored, after which the relevant XML sources need to be queried, to extract the requested information. To solve this problem, several solutions are possible. For the webspace search engine, we have chosen to use the Data eXchange Language (DXL) for this purpose. DXL offers a framework for data exchange, where several heterogeneous sources can be combined into a single target. In Section 4 DXL, and its integration in the Webspace Search Engine is discussed in more detail. In the example of Section 4.5, it is shown how a query is formulated, transformed in DXL and its result is presented.

3.1 Webspace Search Engine

Querying a webspace is done with the Webspace Search Engine. Because the formulation of the query over a webspace is far more complex, than for a general search engine, it is important that this process is carried out using a graphical user interface. The GUI used by the Webspace Search Engine allows a user to formulate a query, by selecting classes and associations from the webspace schema. In Figure 3 the Graphical User Interface (GUI) of the Webspace Search Engine [2] is presented. Formulating a query consists of three steps: (1) Formulating the query skeleton, (2) defining the constraints, (3) determination of the resulting structure.

Formulation the query skeleton

The bottom-left area of the GUI is reserved for the visualization of the webspace schema, once a particular webspace is chosen from the tree in the upper-left area. As can be seen in the figure, the visualization of the webspace schema only contains the class and association concepts defined for a webspace. By selecting concepts from the schema into a single graph, (the skeleton of) a query over a webspace is formulated, without any knowledge of the structure of the documents at the physical level of a webspace. This also implies, that queries can be formulated and constructed, combining conceptual information that is obtained from multiple documents.

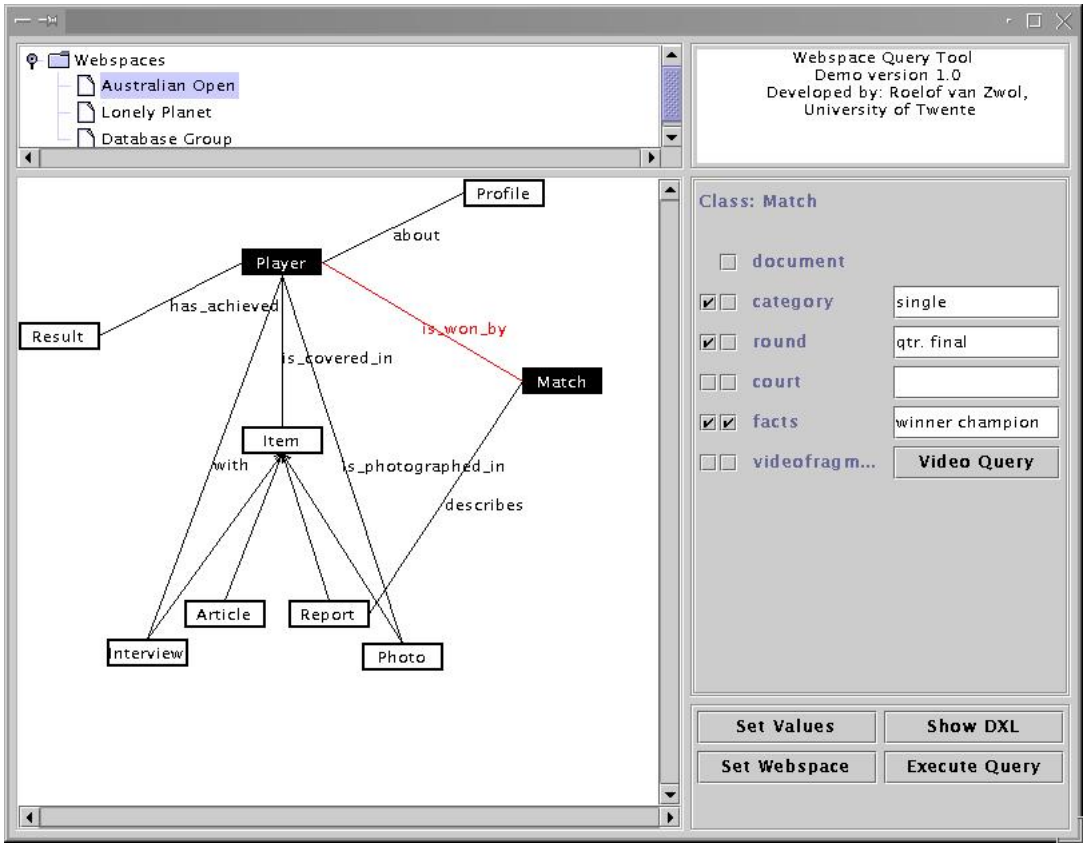

Figure 3: GUI of Webspace Search Engine.

Formulating constraints

Selecting class and association concepts from the schema is the first step towards formulating a query. The second step involves the formulation of constraints. When selecting a concept (class or association), its attributes become visible in the middle-right panel. In front of each attribute two columns of checkboxes are placed. The checkboxes of the first column indicate whether an attribute is used as a constraint of that query. To complete the constraint criteria must be supplied, using the textfield behind an attribute, or in case the type of an attribute is a multimedia class, a more complex interface might be started, to formulate a content-based sub-query for that attribute. In case of a video fragment, a querytool is started, that allows users to define content-based queries over tennis videos[10].

Formulating the resulting structure

The second column indicates whether an attribute should be included into the result of a query. As the result of a query, the Webspace Search Engine returns an XML-document, that contains the concepts selected by the user. When attributes of more than one class are selected into the result, the user is asked to specify which class should form the root of resulting fragment. Figure 8 shows the resulting XML document of a query over the Australian Open webspace.

4 Under the Hood of the Webspace Search Engine

We explained the importance of a graphical interface for query formulation purposes, but even more important is what's beneath the surface. In this section we will discuss how queries are constructed,

executed, and presented, once the query is formulated. For that purpose DXL is used, as will be discussed in the following subsections.

4.1 Data Exchange Language

The main goal of DXL[22] is to provide an extensible framework for data exchange over web-based applications. The exchange format used by these applications may vary from application to application, since a uniform format is still missing, despite all standardization efforts. To bridge this gap DXL offers a framework, which can embed other query languages. This way a particular source is queried by its own query language and the derived intermediate results are exchanged using DXL's framework, also referred to as the DXL base language.

4.2 DXL's features

DXL explores four main features, which are discussed by the items below.

- **Data integration.** This feature of DXL is obvious, since it forms the main focus and motivation of DXL. We already argued for the need to exchange data over web-based applications. DXL allows heterogeneous sources to be queried and the results to be integrated in a single target. The main contribution of DXL to this research area is that it offers a data(type) independent approach for data exchange over multiple sources.

- **Extensible.** To embed other query languages in DXL, DXL's extensibility is crucial. The DXL language base uses two instructions (`query` and `construct`) for this purpose, which must be defined for each extension of DXL, given a specific data type.

- **Template-based.** The behavior of a DXL query is specified in one or more templates. Each template defines a part of the target structure, or simply passes some intermediate results to the next template. Within a template it is only possible to query one source. Thus if multiple sources need to be queried, there are at least as many templates needed.

- **Recursive behavior.** DXL's recursive behavior, i.e. the recursive calls to one or more templates, allows source/target data-types to be queried/constructed recursively. As a result structures can be queried or constructed, having an arbitrarily nested depth in their structure. In case of newsgroups, where messages can have any number of follow ups such a property is very useful [22].

4.3 DXL Language Definition

We already explained that DXL should be seen as a framework, rather than a query language, since its main purpose is to provide a framework for data exchange, instead of querying a particular source. In this section we will not give a formal language definition of DXL, but explain some of the basic instructions by the hand of the example given in Figure 4. It shows the framework provided by DXL, including the syntax of the DXL extensions for querying a Postgres database, and constructing XML documents. The syntax of the DXL base language is specified in XML, but as we can see in the Figure, the query languages embedded by the DXL extensions can maintain their own syntax, as is done for the Postgres extension.

Each query starts with the `dxl_query` instruction, which has two required attributes: `target` and `driver`. The `target` attribute specifies the target location of the data type that is constructed by the query, while the `driver` attribute specifies which DXL extension should be used for the execution of the query. In the example of Figure 4 the target is an XML document, for which the extension "dxl.client.xml" is invoked. All `construct` instructions of a query refer to these two attributes, when constructing the result. Each query must always contain a `template` instruction with the `name` *main*. This template initiates the

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <dxl_query driver="dxl.client.xml" target="result.xml">
03.   <template name="main">
04.     <param name="param1" select="Hello World"/>
05.     <query driver="dxl.client.database.postgres"
06.            source="jdbc:postgresql://pub/ausopen">
07.       select $param1 as result;
08.     </query>
09.     <construct>
10.       <element name="result">
11.         <attribute name="value" value="$result"/>
12.       </element>
13.     </construct>
14.   </template>
15. </dxl_query>
```

Figure 4: Example DXL query.

execution of the query. When a query is executed, the instructions defined in this template are carried out in a sequential order, following a depth-first approach.
Within a `template` instruction, several instructions can be called. The example of Figure 4 starts with a call to the instruction `param` (line 4), which allows us to define a variable, by the `name` *param1* with the `value` "Hello World". Each parameter definition is only valid within that template, unless it is specifically passed to another template. In a later example we will use this feature. Lines 5-8 show the syntax of a `query` instruction. In this case the `source` is a Postgres database. The syntax of the source dependent code is defined within the `driver` extension "dxl.client.database.postgres". Each query instruction requires the attributes `source` and `driver` to be specified, since a single DXL query can query multiple heterogeneous sources. Line 7 of the example shows the SQL-statement that is executed. At execution time, the parameter *param1* is replaced by its value. The result of a `query` instruction is always passed to the following `construct` instruction. From within both the `query` and `construct` instructions calls can be made to instructions belonging to the DXL base language, besides calls to driver dependent instructions (not illustrated here!). The construct instruction uses the instructions `element` and `attribute` to construct the result of the query.
At this point, we end our discussion of the DXL language definition. For a more formal discussion of the DXL language definition, and its extensions we refer to [22]. In the remainder of this article we will present some other instructions, but we feel that these instructions can be explained on a intuitive basis.

The Postgres and XML Extension for DXL

The Webspace Search Engine uses two extensions of DXL. The first extension is the Postgres extension, used to query our object server for the conceptual and multimedia meta-data. This extension is rather simple, and uses (Postgres)SQL directly as its syntax, as can be seen on lines 5-8 of the example query presented in Figure 4. The XML extension, however, uses again XML as its syntax. The extension defines four new instructions. The instructions `element`, `attribute` and `instance_of` are used to construct new XML structures (see lines 41-55 of Figure 7), while the `get` instruction (see lines 37-40 of Figure 7) is used to query XML sources. Like DXL itself, the `get` uses an XPath[15] implementation to traverse through the XML documents.

4.4 Integrating DXL in the Webspace Search Engine

After the previous sections on DXL, it might be clear that DXL is not meant as an end-user (conceptual) language. But, due to its template-based syntax, it is easy to generate. In case of the Webspace Search Engine DXL queries are generated, based on the requirements supplied by the user, through the GUI of the Webspace Search Engine.
The integration of DXL into the Webspace Search Engine uses the framework, presented in Figure 5, to construct the query. As can be seen in the figure, the query is formulated by three templates. The *main* template is always the same, and constructs the root-element '*query*' of the XML-document (*result.xml*). Line 6 shows that a call is then made to the second template, with the name *query* (lines 11-23). This template always contains a `query`-instruction (lines 12-15). This `query` instruction is responsible for gathering the requested meta-data from the object-server. When a query is constructed, the comment of line 14 is replaced by a PostgresSQL-statement, which retrieves the conceptual information, using the constraints supplied by the user. The constraints concern both the conceptual, and multimedia requirements.

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <dxl_query driver="dxl.client.xml" target="result.xml">
03.   <template name="main">
04.     <construct>
05.       <element name="query">
06.         <call_template select="query"/>
07.       </element>
08.     </construct>
09.   </template>
10.
11.   <template name="query">
12.     <query driver="dxl.client.database.postgres"
13.            source="[jdbc-connection to meta-database]">
14.       <!-- Meta-database sub-query -->
15.     </query>
16.     <construct>
17.       <element name="result">
18.         <call_template select="construct-1">
19.           <!-- DXL code to pass parameters to next template -->
20.         </call_template>
21.       </element>
22.     </construct>
23.   </template>
24.
25.   <template name="construct-1">
26.     <query driver="dxl.client.xml" source="[XML-document]">
27.       <!-- XML part of DXL query -->
28.     </query>
29.     <construct>
30.       <!-- Construction of XML fragment for single result -->
31.     </construct>
32.   </template>
33. </dxl_query>
```

Figure 5: DXL's query formulation framework for the Webspace Search Engine

The result of the `query`-instruction is then passed to the construct instruction of lines 16-22. There each result is encapsulated by a "*result*"-tag. From within this tag, a call is made to the template with the

name *construct-1* (lines 25-32). Since parameters are only available within the scope of a template, they need to be passed to the next template explicitly (line 19). Depending on the requested information, i.e. the complexity of the resulting XML-document, one or more templates are needed to compose the result. These templates are named "*construct-n*", with *n* being a variable. The explanation for this is simple: within a template, it is only possible to query one source. In case of the Webspace Search Engine, we will always query the object server. If the requested information can not be obtained from the original XML document directly, we will have to access the original XML-documents directly. Keep in mind that the object server only stored the meta-data involved and not the (multimedia) objects, like the hypertexts, itself. The query instruction of lines 26-28 is used to directly retrieve the requested multimedia information from the original XML-document. Finally the `construct` instruction of lines 29-31 is responsible for the generation of the XML-fragment representing a single result.

4.5 Querying the Australian Open Webspace

In this section we will discuss two queries, which can be formulated with the Webspace Search Engine, based on the Australian Open webspace. Fragments of both queries are presented in 6 , and 7. For the first query all italic code needs to be ignored. These code fragments need to be taken into account. For the second query, which extends the first query.

Query 1

Below a textual formulation of the first query is given, based on a part of the webspace schema, as shown in Figure 2:

> "*Give the name and picture of players, that played a match in the 'quarter finals' round of the 'singles' competition.*"

In Figure 6 the first part of the generated DXL query is given. It shows the template with the name *query*, which yields a part of the framework, as discussed in Section 4.4. The SQL statement of lines 14-19, contains the conceptual requirements, and delivers all the requested information, in this case the name of the player, and the source of the player's picture. Since all information needed to construct the result is obtained from the object server, the next template, *construct-1* is relatively simple. It only consists of a `construct` instruction (lines 41-55) to produce the result.

Query 2

Below the textual formulation of the second query is given, which extends the first query and introduces a multimedia component:

> "*Give the name and picture of players, and facts of the match, where the players played a match in the 'quarter finals' round of the 'singles' competition, the facts contain terms like 'winner' and 'champion*"

Although the extension of the query looks simple, the actual DXL query becomes relatively more complex. The attribute *facts* is of type *Hypertext*, which is a multimedia class. Therefore we will have to access the XML-document, that describes these facts directly, and integrate the requested information into the result of the query. Lines 14 and 23 of Figure 6 show that the conceptual query is also changed. The query now includes a multimedia component, since the facts are described in a hypertext. Furthermore, the result of the conceptual query (line 14) delivers a ranking of the result, and a *url*, which points to the XML-document that needs to be accessed. Also very interesting are lines 37-40 of Figure 7, which show the `query`-instruction that is executed to obtain the requested facts. The `instance_of` instruction imports the XML-fragment, containing the requested facts, directly into the result.

The final result of this query is shown in Figure 8. The same style-sheets, that are used to visualize the original XML-documents, are reused to present the result of the query.

```
11.   <template name="query">
12.     <query driver="dxl.client.database.postgres"
13.            source="jdbc:postgresql://pub/ausopen">
14.       select name, source, url, sum(tfidf) as rank
15.       from player, match, image, is_played_by, documents, ht_tfidf
16.       where match.category~*'single' and match.round~*'Qtr. Final'
17.         and match.id=is_played_by.match
18.         and player.id=is_played_by.player
19.         and player.picture=image.id
20.         and match.documents[1]=documents.id
21.         and match.facts=ht_tfidf.hypertext
22.         and (term='winner' or term='champion')
23.       group by name,source,url order by rank desc;
24.     </query>
25.     <construct>
26.       <element name="result">
27.         <attribute name="rank" value="$rank"/>
28.         <call_template select="construct-1">
29.           <with_param name="name" select="$name"/>
30.           <with_param name="source" select="$source"/>
31.           <with_param name="url" select="$url"/>
32.         </call_template>
33.       </element>
34.    </construct>
35.  </template>
```

Figure 6: Part 1 of the example queries.

5 Conclusions and Future Work

The Webspace Method, and in particular the Webspace Search Engine, as presented in this article introduces a whole new approach to search through, or more specific, to formulate queries over a collection of web-based documents. Based on conceptual modeling, the Webspace Method introduces a semantical level of abstraction, by defining and modeling concepts, that describe the content of a webspace in an object-oriented schema, called the webspace schema. This schema forms the key ingredient for querying a webspace. Rather than querying the content of a single document at a time, a webspace is queried as a whole, by its schema. Furthermore we have shown that the result of a query over a webspace is not restricted to the retrieval of a collection of URLs pointing to relevant documents, but that the result can be formed by reconstructing a combination of concepts, earlier defined for a webspace. Finally we have discussed how DXL is integrated into the Webspace Search Engine, and how DXL queries are generated, to construct the result.

Future Work

For our future work, we plan to investigate the scalability of the Webspace Method, in order to handle larger domains of the Internet. Based on the promising results of a small scale retrieval performance experiment [21], we also intend to carry out more experiments, to measure the contribution of the Webspace Method to the area of information retrieval.

References

[1] R Baeza-Yates and B Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999.

```
36.   <template name="construct-1">
37.     <query driver="dxl.client.xml"
38.            source="$url">
39.       <get from="//match" select="facts"/>
40.     </query>
41.     <construct>
42.       <element name="player">
43.         <attribute name="name" value="$name"/>
44.         <element name="picture">
45.           <element name="image">
46.             <attribute name="source" value="$source"/>
47.           </element>
48.         </element>
49.         <element name="played_by">
50.           <element name="match">
51.             <instance_of select="$facts"/>
52.           </element>
53.         </element>
54.       </element>
55.     </construct>
56.   </template>
```

Figure 7: Part 2 of the example queries.

[2] H.E. Blok, M. Windhouwer, R. van Zwol, M. Petkovic, P.M.G. Apers, M.L. Kersten, and Jonker W. Flexible and scalable digital library search. In *Proceedings of the twenty-seventh International Conference on Very large Data Bases (VLDB'2001)*, Rome, Italy, September 2001.

[3] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. Xml-gl: a graphical language for querying and restructuring XML documents. In *proceedings of the International World Wide Web Conference (WWW)*, pages 1171–1187, 1999.

[4] A. Deutsch, M. F. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for XML. In *proceedings of the International World Wide Web Conference (WWW)*, pages 1155–1169, 1999.

[5] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the boat with Strudel: Experiences with a web-site management system. In *proceedings of ACM SIGMOD Conference on Management of Data*, Seattle, WA, 1997.

[6] N. Fuhr and K. Grossjohann. Xirql – an extension of xql for information retrieval. In *Proceedings of the ACN SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece, July 2000.

[7] D.A. Grossman and O. Frieder. *Information Retrieval: Algorithms and Heuristics.* Kluwer international series in engineering and computer science. Kluwer Academic Publishers, 1998.

[8] Y. Hayashi, J. Tomita, and G. Kikui. Searching text-rich xml documents with relevance ranking. In *Proceedings of the ACN SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece, July 2000.

[9] Z. Lacroix. Retrieving and extracting web data with search views and an xml engine. In *International Workshop on Data Integration over the Web in conjunction with the 13th International Conference on Advanced Information Systems Engineering (CAiSE)*, Interlaken, Switzerland, June 2001.

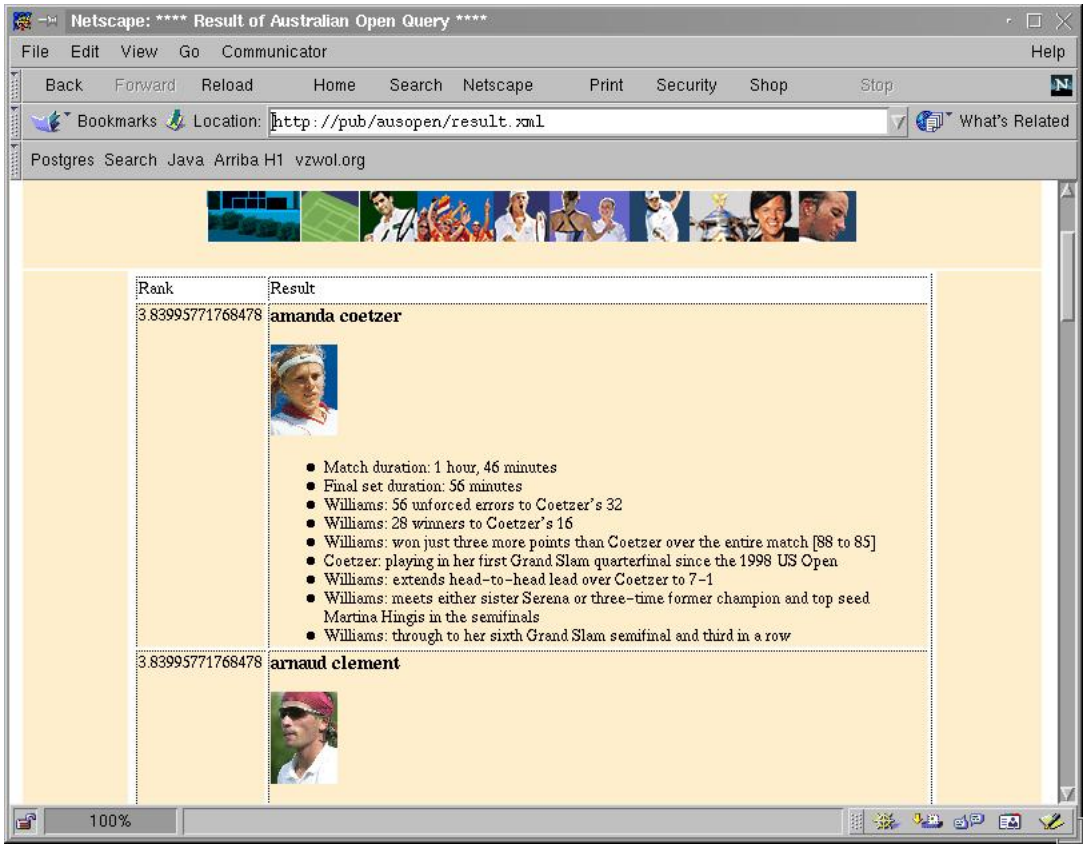

Figure 8: Result of query 2.

[10] M. M. Petkovic and W. Jonker. Content-based retrieval of spatio-temporal video events. In *Multimedia Computing and Information Management Track of IRMA International Conference*, Toronto, Canada, May 2001.

[11] G. Mecca, P. Merialdo, P. Atzeni, and V. Crescenzi. The Araneus guide to web-site development. Technical report, Dipartimento di Informatica e Automazione, Universita' di Roma Tre, March 1999.

[12] Abiteboul S. On views and xml. *symposium on Principles of Database Systems (PODS'99)*, May 1999.

[13] D. Suciu. Semi structured data and xml. In *Proceedings of International Conference on Foundations of Data Organization (FODO98)*, 1998.

[14] A.P. de Vries. *Content and multimedia database management systems*. PhD thesis, University of Twente, Enschede, The Netherlands, December 1999.

[15] W3C. Extensible markup language (XML). Technical report, World Wide Web Consortium (W3C), February 1998.

[16] W3C. XQuery: A query language for XML. Technical report, World Wide Web Consortium (W3C), February 2001.

[17] M.A. Windhouwer, A.R. Schmidt, and M.L. Kersten. Acoi: A system for indexing multimedia objects. In *International Workshop on Information Integration and Web-based Applications and Services*, Yogyakarta, Indonesia, November 1999.

[18] R. van Zwol and P.M.G. Apers. Modelling the webspace of an intranet. In *proceeding of 1st international conference on Web Information Systems Engineering (WISE00)*, Hong Kong, June 2000.

[19] R. van Zwol and P.M.G. Apers. Using webspaces to model document collections on the web. In *proceedings of WWW and Conceptual Modelling(WCM00), in conjunction with ER2000*, Salt Lake City, October 2000.

[20] R. van Zwol and P.M.G. Apers. The webspace method: On the integration of database technology with information retrieval. In *proceeding of Ninth International Conference on Information and Knowledge Management(CIKM00)*, Washington DC., USA, November 2000.

[21] R. van Zwol and P.M.G. Apers. Retrieval evaluation experiment of the webspace method. In *submitted for publication*, 2001.

[22] R. van Zwol, V. Jeronimus, and M. Fokkinga. Dxl: a data exchange language. In *submitted for publication*, 2001.