EVA 2004 - London, 26-30 July 2004 (pages 25.1-25.10) - ISBN 0-9543146-4-6
Document URI: http://www.w3c.it/papers/eva2004-query.pdf

QUERYING XML DOCUMENT COLLECTIONS

Oreste Signore* – Marco Andreini** – Cristian Lucchesi** – Silvia Martelli*
W3C Office in Italy at C.N.R.
(*) Istituto di Scienza e Tecnologie dell'informazione "A. Faedo" (ISTI)
(**) Istituto di Informatica e Telematica (IIT)

Area della Ricerca CNR - Via G. Moruzzi, 1 - 56124 Pisa - Italy
Email:oreste@w3.org
URL: http://www.weblab.isti.cnr.it/projects/QH/docs/

In this paper we describe a query interface towards XML document collections. External schema annotation in RDF contains information used to dynamically build the interface tailored to the user's characteristics and to the document structure, as described by its XML Schema. The interface makes the user aware of structure semantics, so supporting her/him in formulating semantically correct queries. In preparing the query, the user can have access to ontologies or linguistic resources via web services. Queries are prepared in an intermediate format that can be translated into different search engines. The architecture is fully compliant with web design principles and standards.

INTRODUCTION

Querying XML document collections is becoming an important issue ([3], [5], [6], [7], [8], [9], [10]). As XML documents have a structure, users like to understand the semantics of tags and structure ([11]). However, formulating a query is quite complex, and syntax of XPath and XQuery is not best suited for end users.

In this paper[1] we present a query interface that is tailored to the user needs according to some semantic information added to the XML Schema describing the XML documents. The interface helps the user in preparing the query, and is independent from the specific search engine used to query the document base.

In the rest of the paper, we will first discuss some general issues about user requirements. Afterwards, we present the rationale for the user interface and the strategy for query composition. In the third section, we describe the architecture. Fourth section reports some implementation details. Finally, we give some conclusion and plans for future work.

USER REQUIREMENTS

Document searching is mainly addressing search by content, looking for some specific words contained in the documents, while ignoring their structural component. However, some well known drawbacks negatively affect *Precision* and *Recall* factors. The main one is that searching for a word is not searching for a specific concept, as a single word has poor semantics. Just to make a simple example, when searching in a file of documents containing person addresses, the same word can be a person's last name, or can appear in the street or in the location fields. Therefore, documents are often structured in parts, so that users can search terms in specific document's components, reducing the risk of false hits.

[1] This work has been financed by the project QUESTION-HOW (Quality Engineering Solutions via Tools, Information and Outreach for the New Highly-enriched Offerings from W3C: Evolving the Web in Europe), contract IST-2000-28767

To get the goal of looking for the appropriate terms and concepts, it is essential to share the same knowledge base between user and indexer. This is usually done by having access to thesauri or dictionaries of terms, where the user can check the allowed terms and their semantic relationships. Sometimes, a description of the document is available, but in most cases the semantics of the fields is just left to their names, often unclear or misleading. Even more complex is the case of more sophisticated structured documents, where semantics is conveyed by both the field and the *structure*.

```
<books>
  <book number="1">
    <metadata>
      <author>Oreste Signore</author>
      <title>The evolving Web</title>
    </metadata>
    <content>
    <introduction>
         <author>Oreste Signore</author>
      </introduction>
      <part number="1">
        <chapter>
          <author>Chris Green</author>
          <author>Oreste Signore</author>
          <author>John White</author>
          <title>About the future of the Web</title>
        </chapter>
      </part>
      <part number="2">
        <chapter>
          <author>John Smith</author>
          <author>Chris Green</author>
          <author>Jim Brown</author>
          <title>The future of technologies</title>
        </chapter>
      </part>
    </content>
  </book>
</books>
```

Figure 1 - A sample XML document

Just as a trivial example, think to a simple document representing a scientific book (Figure 1). In this case, the tag `<author>` conveys the semantics that *Oreste Signore* is the author of some parts of the book, and the same is for *Chris Green* and *John White*. However, their order can convey additional information: *Chris Green* is the first author of the chapter whose title is "*About the future of the Web*". Therefore, the user can be interested in searching for books containing *chapters* written by *Oreste Signore AND Chris Green*, where *Chris Green* is the *first* author, and there are *no more than three authors*.

In passing, it is worthwhile to note that the tag author has no special meaning by itself, as the user is supposed to be aware that it refers to the author of the paper. To have a really effective search, the user should be aware of the meaning of the tag, e.g. that its semantics is the same as the `<dc:creator>` tag as defined by the Dublin Core initiative ([4]). Hence, we need a *semantic description* of every tag and a clear understanding of the *structure* of the searched documents.

In the World Wide Web users are often searching on multiple databases or document collections, having different tag names and structures, even if containing intrinsically homogeneous or strictly related documents (like books, papers, laws, sentences). Having the possibility of identifying semantic equivalences in different document

collections, and hence having the possibility of searching and linking documents in these collections, is a relevant issue in the Semantic Web panorama ([15])

DESIGNING A QUERY INTERFACE

The main objective of the query interface is to support users in formulating queries that impose conditions on both elements content and document structure. Main goal is hence to support the user in formulating semantically correct queries, avoiding the burden of using complex languages ([1]). A *semantically correct query* is a query that specifies consistent conditions both on the content of the elements, as well as on the structure. Needless to say, a semantically correct query can return an empty result set, but this will not be the effect of specifying unacceptable conditions.

To this aim, the user must be supported for:

- understanding the *elements semantics* and the *document structure*;
- inserting *correct values* when specifying condition on elements;
- identifying appropriate *concepts* to search for.

Regarding the first issue, we just recall that element's semantics can't simply rely on the tag name, which is often esoteric or meaningless. In XML Schema it is possible to use a description, but in most cases designers leave it empty, if they even design a schema.

To insert correct values when specifying conditions, it is necessary to be aware of *allowed values* and *constraints*. Again, at XML Schema level it is possible to specify lists of allowed values for the elements. However, it is not a common practice, and the lists are fairly static and difficult to share. In addition, designers can't specify constraints when allowed values for an element depend on the values taken by another element. For example, take the case where we have a single schema describing heterogeneous documents, like *books* and *conference proceedings*. It is evident that, in this case, the element `<conferenceLocation>` must be null if `<documentType>` takes the value "*book*", while is mandatory when it takes the value "*conference Proceedings*".

Finally, there are some semantically rich elements, like keywords, where finding the appropriate *concept* to search for is of crucial importance, and requires sharing the same knowledge base between user and indexer. This can be achieved sharing the access to the thesaurus, or, more generally, to the ontology the element is referring to. The same applies when the user is interested in accessing a free text element, whose searching is much more effective if (s)he can have access to one or more thesauri showing semantic relationships among concepts that can be found in the element.

Furthermore, users need support in *interaction*. First of all, descriptions of the elements, commands and help must be provided in *multiple languages*. But one more aspect, that is implicit in the Web, is the ability to cope with *different cultures* of the users. Users can also differ in *level of expertise*. A casual user will need extensive help and guidance, while expert user will be aware of structure and tag semantics, and will prefer a more concise interface, also suitable if using a slow connection or limited capabilities devices (e.g. a PDA)

Querying would also be possible using XPath expressions or XQuery, but their syntax is of difficult use for any end user.

Finally, user interface is enriched if returned document are processed, both for the traditional emphasizing of search terms or formatting, as well to enrich documents with hyperlinks, annotations, additional info derived from an analysis of the document.

A few experiences are as frustrating as querying. Even if the user is aware of the document structure and element semantics, (s)he will have to pass through a long series of steps, selecting fields, Boolean operators, parentheses, etc.. And, finally, (s)he realizes that (s)he wasted a lot of time to prepare a very simple query, while complex queries are often impossible to state. This kind of problems is emphasized when facing complex structures as XML documents are.

We tried to find a way of composing the query which is absolutely general, and does not depend on the specific search engine, but only on the supported functionalities.

The query is composed by navigating the document structure and interacting via a Query Form, which is set up using information contained in the XML Schema and its external annotation in RDF.

Querying hierarchical structures, as XML documents are, we are faced with the problem known as normalization in the era of hierarchical DBMSs. Essentially, when we specify conditions on independent elements, belonging to different paths, we have to specify the common ancestor of the elements which identifies the referenced sub-tree. For example, referring to a document collection containing records like the one in Figure 1, a query imposing conditions:

author=”Oreste Signore” AND title=”The future of technologies”

must specify the root (`<chapter>` or `<content>`) of the sub-tree where the two conditions have to be satisfied. It is natural, for the user, to associate the identification of the sub-tree root with a navigation on the document structure, during which elementary conditions on the element values are imposed.

On this basis, and also considering that composing a complex query can result in a long interaction, while expert users would prefer to formulate the simple condition once, and subsequently reuse these elementary components combining them with Boolean operators and parentheses, we opted for a solution where the query is composed by a combination of several items.

A **query** (identified as q1, ..., q99) is a boolean composition (with parentheses) of one or more *queryFragments*. A **queryFragment** is a boolean composition (with parentheses) of *queryTokens*. The queryFragment is constructed automatically in a navigation upon the documentTree, where user specifies conditions on the elements. By default, the queryTokens are ANDed. The **queryToken** is the most elementary component of the query. A queryToken is built by a single interaction with the queryForm (Figure 4). Would the user like to specify a condition upon several instances of an element, (s)he will initiate a new interaction upon the same element.

For example, supposing to have a document (as in Figure 1) where the element `<author>` contains just one of the authors, and is therefore repeated as many times as the authors are, to search for papers having as `<author>` "X" AND "Y", the user will have to specify two query tokens: `<author>="X" AND <author>="Y"`, while `<author>= "X" AND "Y"` will be meaningful only if the element `<author>` contains all the authors. What will be the right way to formulate the query, will be clear by the awareness of the document structure and content, as explained by the longDescription property. Every queryToken is identified as: qT1, ..., qT99 in a single qF.

THE ARCHITECTURE

The system architecture was conceived to be as open as possible, leaving the possibility of interfacing any kind of XML document collection, without imposing any manipulation of the schema, which is externally annotated. Some processsing occurs

invoking services, identified by URIs. The architecture is fully compliant with Web standards and design principles, namely extensibility, interoperability and decentralization. A rough sketch of the architecture is in Figure 2.

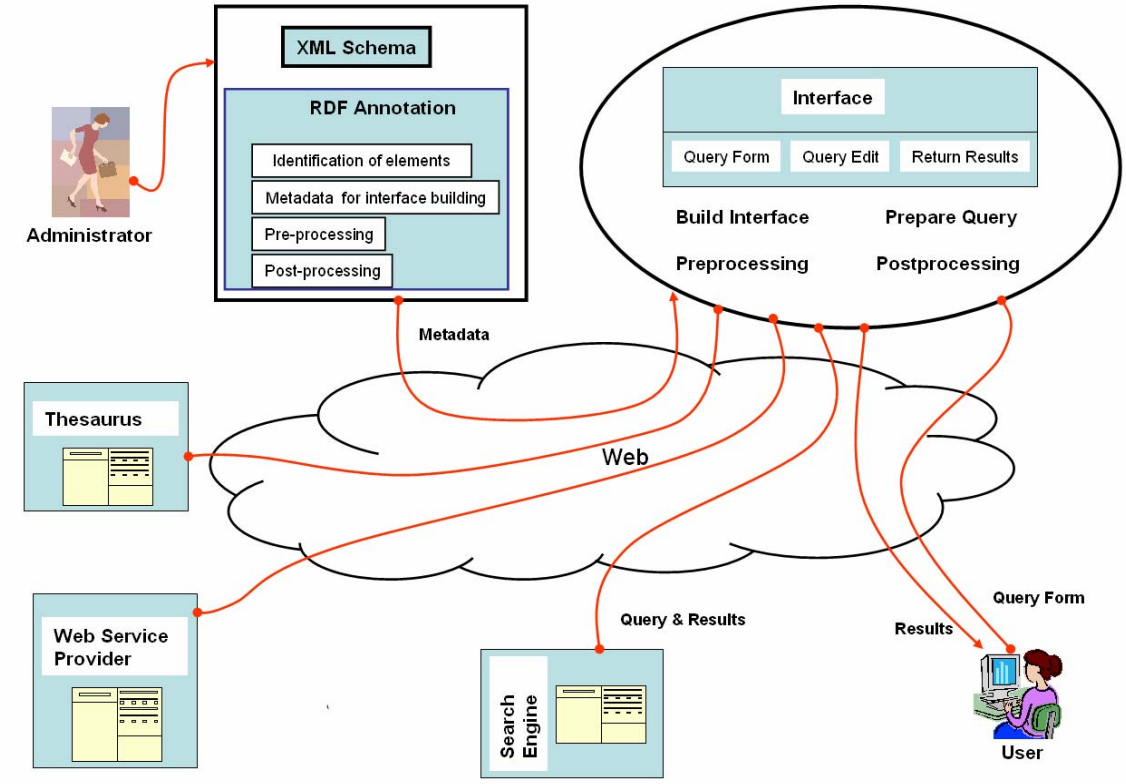

Figure 2 - The overall conceptual architecture

Functionally, we can distinguish three levels: the collection *administrator*, the *user*, the *document server* (including the search engine). All of them will be here briefly sketched. For a more complete description the reader is referred to [12].

The *administrator*, who is supposed to have an in depth knowledge of the document collection, is in charge of *managing* the collection, using appropriate indexing schema and search engine. It is responsibility of the administrator to build the XML Schema, if it doesn't exist yet, and provide semantic description of XML Schema elements as external annotation in RDF. Metadata can be imported in the system, and stored in its internal objects. On the other hand, the system provides its own interface to add metadata to the system objects. Such metadata can be subsequently exported as a RDF annotation. In such a way, it is possible to create the RDF annotation without making use of any ad hoc editor.

A *User profile* is needed to personalize the *level of detail* of information supplied at the user interface level (novice or expert user), the *interaction language*, and the *presentation* of returned documents. In the present implementation, the User Profile has been kept as simple as possible, leaving room for subsequent improvements. The preferred language and other characteristics can be dynamically varied by the user, and have an immediate effect.

Element metadata are essential to describe data and tailor user interface and system behaviour. Some of them can be derived by the XML Schema, while others can only be specified by an expert of the domain, and have no way to be properly coded in the

schema itself. In addition, we remember that a basic design issue has been to leave the XML Schema unchanged. From the XMLSchema we can take some properties of elements and attributes, like range, multiplicity, pattern, list of admitted values, content type (text, element, mixed). We can have some additional metadata to describe properties of elements and attributes in a RDF file. The RDF should contain information to univocally *identify* elements and attributes, descriptions to help users to understand their *meaning*, *transformation rules* (e.g. dates can be stored in Gregorian calendar, but queried and displayed in a different calendar, like muslin or Jewish), kind of *control*, if any (list of values, controlled vocabularies, ontologies, etc.).

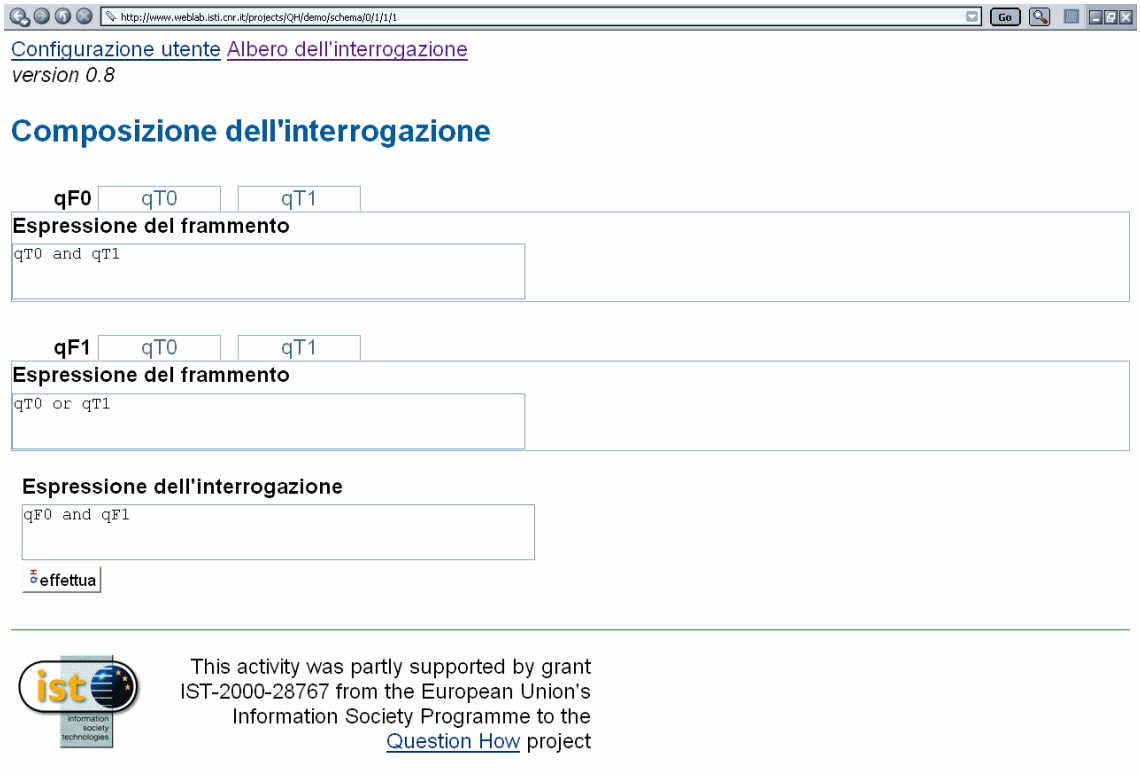

Figure 3 - Query editing (in Italian)

Metadata information is used both in the personalization of the interface (element properties and description, and pre-processing and post-processing), and in supporting the interaction (constraints, controlled content elements). To avoid excessive complexity in stating the constraint using RDF, they are described using natural language expressions. In a future release, the possibility of stating in a purely declarative way some simple and common constraints (e.g. existence constraints, elements whose values must be in a well defined equivalence relation, etc.) will be considered. For element whose content is controlled by list of values, dictionaries, thesauri, we can have automatic or user activated (Figure 4) invocation of web services or CGI applications, as stated in the RDF annotation.

The *user interface* is dynamically created on the basis of information managed by the Administrator, and is capable of helping the user in expressing all the queries (s)he is interested in, assuring their correctness and supplying almost all the features supported by the search engine in the background. For each element, the system is well aware of its semantics and constraints, which can be shown to the user.

The user can navigate the structure and prepare the query. The query is composed in a fully *general format*, which can subsequently be mapped on the specific search engine.

Metadata are used by the system to produce the **Query Form**, that will contain all the information needed by the user agent (the browser in a first implementation) to implement the User Interface. This means that the Query Form will contain all the constraints applicable to the single elements, as well as the information needed to enable the user to formulate semantically correct queries.

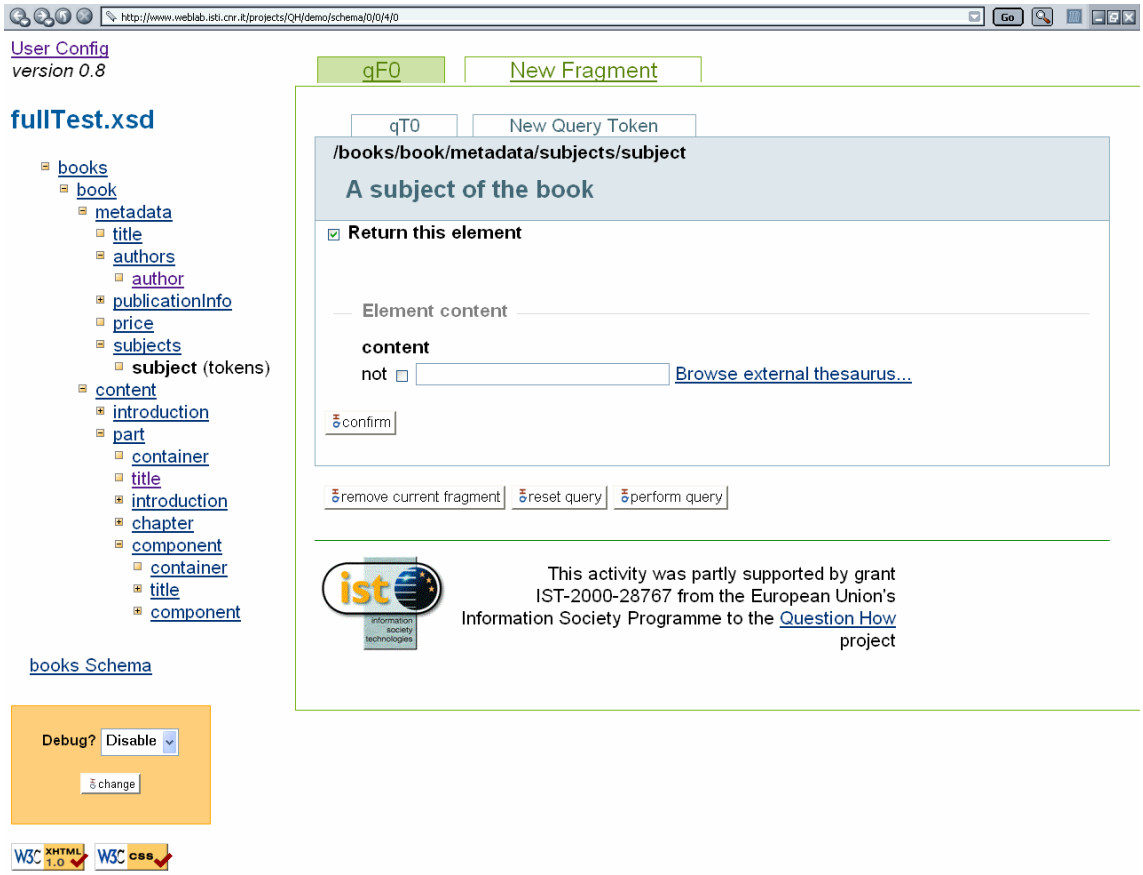

Figure 4 - Filling a queryToken (in English)

The user interface implements the human computer interaction. This implies:

- presenting a query form
- displaying the document structure
- managing the interaction with additional information sources (dictionaries, thesauri, ontologies, ...)
- composing the query in a form suitable to be processed by the underlying search engine. In this implementation we support a subset of XQuery language with some of the features available in IR query languages (proximity, regular expressions, ...)
- editing and submitting the query
- on returned documents, displaying them.

The first step in using the system is to select the document collection and selecting the User Profile. Subsequently (Figure 4), the user will get, on the left of the screen, the document tree, with nodes that can be expanded or compressed. For each node, a click

will activate, on the right part of the screen, the **Query Form**, where the user can enter the desired values and operand to build a **queryToken**. In composing the queryToken, user can select/deselect the element as a returned element, asking for support from a dictionary or thesaurus. By default, the queryTokens filled in a single navigation are ANDed and composed into a **queryFragment**, which, in turn, can be combined with other queryFragments to build a query. Every queryFragment is identified as: qF1, ..., qF99. A queryFragment is started when user selects: **newFragmen**t (so starting a new navigation upon the documentTree).User can *edit* queryFragments and query, adding parentheses and inserting or modifying boolean operators (Figure 3) to achieve different combinations of queryTokens. To modify a queryToken, user must go back to the token and interact via the query Form. By default, queryFragments are ANDed. A query is produced when the user specifies: **buildQuery**.

At the *Document Server* level we have the process of converting the incoming query into the format supported by the search engine, executing the query, and returning the document set.

IMPLEMENTATION

In the current implementation we adopted, as search engine, the *XCDE library*, developed by the Department of Computer Science of University of Pisa (prof. Paolo Ferragina). During this project, the library has been further developed, to support a sophisticated query language combining some characteristics of XQuery ([17]) with most of the IR functionalities detailed in [18]) as well other powerful string-based queries, like regular expressions and error matches.

The library provides an API with a rich set of C functions to operate on its whole collection of data structures and algorithms. It may implement most of the basic functionalities of XQuery, and support more complex IR-like searches. The aim of its design is to manage efficiently and effectively XML documents that contain significant textual parts and/or a deeply nested hierarchical structure, and whose attribute values are complicated strings of numbers and letters. This is the typical scenario encountered in literary texts tagged via XML-TEI.

The query syntax is similar to SQL: SELECT-FROM-RETURN, but here the SELECT clause is specified by means of an XML piece of well-formed text. The output of the query (the snippet) can be formatted within the RETURN clause which, again, includes an XML piece of well-formed text. A special attribute (called pivot) whose name is xml_var is added to SELECT clause to identify the sub-trees satisfying the query.

The implementation environment is fully based on open source software, (Zope, Python, Soaplib Python, parsedXML). Postgresql is supporting storage of dictionaries and thesauri. The present version depends on parsedXML, using its DOM functions to access the XML schema. Next releases will make use of SAX functions embedded in Python.

Many of the components have been fully implemented. Among them, we list:

- A parser to get a simplified XMLSchema.
- A transformer from XMLSchema to a XHTML tree, allowing the navigation upon the tree.
- The administration interface to store the Element Metadata.
- A form builder based upon the processed XMLSchema and Element Metadata.
- A modular component to assemble the query, edit queryFragments and the final query, and translate the query in XCDE compliant format.

- The wrapper to call XCDE Library functions and getting query results.
- A sample web service to check query terms against a dictionary.

As *case study* we used a little variant ([2]) of the document collection referred by [18]. The RDF annotation ([13]) of its schema ([14]) specifies several types of constraints:

- local list (values specified in the RDF annotation itself)
- external list (invoked via a web service)
- thesaurus (invoked as an external application)
- elements in relationship.

A demo is available at http://www.weblab.isti.cnr.it/projects/QH/demo/.

CONCLUSION AND FUTURE WORK

We developed a consistent framework for intelligent and controlled access to XML document collections. The document collection schema has been left unchanged, but enriched with an external annotation in RDF.

The administrator of the document collection can refer any external knowledge source, invoking external thesauri via web services (if such services are available) or via a traditional CGI application. How to invoke the service or the application is specified in an appropriate XML resource, identified by its URI.

The interface is automatically tailored to any XML document collection described by an appropriate XML Schema, and can be tailored to any search engine. Presently, no facilities are provided to state functions and search operators supported by the search engine. Some of the features have not yet implemented, but they can be classified as "conventional programming" with no special difficulties.

As a future work, we are considering the possibility of using this interface for editing of XML documents, and the migration towards a client oriented architecture, making use of the facilities provided by XForms.

A further direction is to extend the search towards heterogeneous document collections, but comprehending semantics of the document structure is a prerequisite that will need further investigation and effort.

ACKNOWLEDGEMENTS

Authors wish to acknowledge prof. Paolo Ferragina (Computer Science Department of Pisa University), and his staff (namely Alessandro Perucci and Rosanna Rossi).

REFERENCES

[1] Angela Bonifati, Stefano Ceri: Comparative Analysis off Five XML Query Languages, *ACM SIGMOD Record, **29**(1): 68-79 (2000)*

[2] http://www.weblab.isti.cnr.it/projects/QH/docs/books.xml

[3] Stefano Ceri, Piero Fraternali, and Stefano Paraboschi. XML: Current Developments and Future Challenges for the Database Community. *Proceedings of the 7th International Conference on Extending database Technology (EDBT 2000), pp. 3-17, Konstanz, Germany (2000)*

[4] The Dublin Core Home Page, URL: http://dublincore.org/

[5] Alin Deutsch, Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, David Maier, Dan Suciu. Querying XML Data. *IEEE Data Engineering Bulletin, **22**(3):10-18 (1999)*

[6] R. Guha , Rob McCool , Eric Miller. Semantic Search. *Proceedings of WWW22003, The Twelfth International World Wide Web Conference, 20-24 May 2003, Budapest, Hungary, pp 700-709*

[7] Alon Y. Halevy, Zachary G. Ives, Peter Mork, Igor Tatarinov. Piazza: Data Management Infrastructure for Semantic Web Applications. *Proceedings of WWW22003, The Twelfth International World Wide Web Conference, 20-24 May 2003, Budapest, Hungary, pp.556-567*

[8] Hayashi, Y., Tomita J., Kikui G.: Searching Text-rich XML Documents with Relevance Ranking. *ACM SIGIR2000 Workshop on XML and Information Retrieval, July 28, 2000, Athens, Greece*

[9] Kotsakis E: Structured information retrieval in XML documents. *Proceedings of the 2002 ACM symposium on Applied Computing, Madrid, Spain, pp. 663-667, ISBN 1-581113-445-2*

[10] Miller, J.A.; Sheth, S.; Querying XML documents. *Potentials, IEEE, Volume: 19, Issue: 1 , Feb.-March 2000 Pages:24 – 26*

[11] Gonzalo Navarro and Ricardo Baeza-Yates. Proximal nodes: a model to query document databases by content and structure. *ACM Transactions on Information Systems, **15**(4): 400-435 (Oct. 1997)*

[12] http://www.weblab.isti.cnr.it/projects/QH/docs/deliverable.html

[13] http://www.weblab.isti.cnr.it/projects/QH/docs/fullText.rdf

[14] http://www.weblab.isti.cnr.it/projects/QH/docs/fullText.xsd

[15] http://www.semanticweb.org/

[16] http://butirro.di.unipi.it/~ferrax/xcde/xcdelib.html

[17] XQuery 1.0: An XML Query Language - W3C Working Draft 22 August 2003, http://www.w3.org/TR/xquery/

[18] XQuery and XPath Full-Text Requirements - W3C Working Draft 02 May 2003, http://www.w3.org/TR/xquery-full-text-requirements/