Querying Structured XML Document Collections

Oreste Signore* – Marco Andreini** – Cristian Lucchesi** – Silvia Martelli*

(*) W3C Office in Italy at CNR – ISTI    (**) now at CNR - IIT
Area della Ricerca di Pisa San Cataldo - Via G. Moruzzi, 1 - 56124 Pisa
Email: oreste@w3.org

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

1 Introduction

Querying XML document collections is becoming an important issue [1], [2], [3], [4], [5], [6], [7], [8]. As XML documents have a structure, users like to understand the semantics of tags and structure [9]. 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 supports 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 fourth section, we describe the architecture. Fifth section reports details about implementation and case study. Finally, we give some plans for future work.

2 General Issues about User Requirements

Users are concerned with the three essential document components: content, structure and presentation. Content is the most widely addressed aspect. All traditional Information Retrieval Systems, and also many search engines, perform querying looking

\(^1\) 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
for some specific words contained in the document. This is often referred as search by content. However, there are some well known drawbacks, which negatively affect Precision and Recall factors. The main one is that searching for a word is not searching for a specific concept. In fact, a single word has poor semantics, as we can have many cases of a single word having different meanings, as well as we can be faced with the case where the same word can be significant only if it appears in a specific context. Just to make a simple example, think to a search in a file of documents containing person addresses. The same word, that in the user's mind is relevant to select a person identified by his/her last name, can appear in the street or in the location fields. Therefore, not surprisingly, documents are often structured in parts, so that users can search terms in specific document's components, reducing the risk of false hits.

Fig. 1. A sample XML document

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 name: this is unacceptable in a truly World Wide Web environment. Even more complex is the case when we are faced with more sophisticated structured documents. In this case, that is becoming more and more common with the wide adoption of XML,
semantics is conveyed both by the field and by the structure. Therefore, knowing the
document structure is an essential need.

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 [10]. Hence, we need a semantic description of every tag and a clear
understanding of the structure of the searched documents.

As the Web is a World Wide Web, it is quite frequent that users are searching on
multiple databases or document collections. As an obvious consequence of the intrin-
sic decentralized nature of the of the Web, we must expect that different document
collections will have different tag names, as well as different structures, even if con-
taining intrinsically homogeneous documents (like books, papers, laws, cultural her-
tage cataloguing cards). Document collections containing heterogeneous, but some-
how related documents (e.g. laws, sentences, regulations) will have different struc-
tures, containing elements having the same semantics, but different names. 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

3 Designing a Query Interface

3.1 The Rationale

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 [12]. 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 need not necessarily return a
document set, it may happen that the result set is empty, but this is different from the
case when the returned document set is empty because user specified unacceptable
conditions (e.g. a condition like: "PY=02" instead of "PY=2002", where PY is the
Publication Year, coded, in the document collection, as a four digits number).

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, would we have a single schema describing heterogeneous documents, like books and conference proceedings, 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. For this specific aspect we recall that effective queries require that user and indexer share the same knowledge base, and 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, multilingual descriptions, commands and help must be provided. But an additional aspect, implicit in the Web, is the ability to cope with different cultures of the users. Just as an example, there are many different ways to express dates, and users can find the format of the specific document collection not the most suitable. 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 facilities device (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.

### 3.2 Query Composition

A few experiences are frustrating as querying. Even if the user is aware of 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 more complex structures as XML documents are. Similar considerations are in [1], which illustrates XQBE, a visual dialect of XQuery. XQBE is a user friendly interface, implementing a visual query language based on the use of trees, coherently with the XML data model.
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. We were well aware that the expressive power of any interface is limited in comparison with textual XQuery, but we agree with [1], thinking that a too complex query interface would anyway fail in replacing the textual query language, while could become fairly too complex for most users.

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 DBMS. 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:

\[
\text{author} = \text{"Oreste Signore" AND title} = \text{"The future of technologies"}
\]

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 single 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 simple conditions 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 query Fragments. A queryFragment is a Boolean composition (with parentheses) of queryTokens. The queryFragment is constructed automatically in a single navigation upon the documentTree, where user specifies conditions on the elements. Every queryFragment is identified as: qF1, ..., qF99. 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.

In the previous examples, we just showed elementary queries, which make use only of the “=” and “AND” operators. Quite obviously, the interface can support much more complex queries, both in terms of operands and operators. More precisely, comparison operators include all the operators used in Information Retrieval Systems (adjacency, phrase, truncation, masking, etc). User can edit the query, inserting an
arbitrary number of parenthesis levels, to formulate complex queries. The rationale was to give adequate support to the user in formulating the elementary tokens of the query (supporting him/her with semantic information and description of the document tree), while leaving the possibility of combining the tokens to support more sophisticated queries. The hypothesis is that any user who intends to formulate a very complex query is certainly able to use the Boolean operators and parentheses.

4 The Architecture

4.1 General Overview

The system architecture is intended 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 processing requires invoking services identified by URIs. The architecture is fully compliant with Web standards and design principles, namely extensibility, interoperability and decentralization.

![Diagram](image)

Fig. 2. The overall conceptual architecture

A rough sketch of the architecture is in Figure 2, which implicitly assumes that the Document Server resides on the same machine where the Search Engine is, as is usual. However, this is not a restriction, as the possibility of having the Document Server
and the Search Engine allocated on different machines depends only on their characteristics. The user interface must just be aware of the location of the Search Engine.

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 [13].

It is worthwhile to stress that it is just the architecture, which is fully distributed, that makes the proposed interface different from so many yet existing “fill in the form” interfaces. The key is that any component can reside on any machine addressable over the Internet. Resources used by the interface are uniquely identified by their URI.

**Fig. 3.** Query editing (Italian language selected in the User Profile)

### 4.2 Administration Level

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.

To build an effective user interface, we must consider two issues. On one hand, the interface must be tailored to the user needs and characteristics, which can be specified
in an appropriate User Profile. On the other hand, characteristics of the elements must be described by appropriate Element Metadata.

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. Defining and maintaining the User Profile is an additional responsibility of the Administrator.

An example of the effect of different values specified in User Profile is seen in Figure 3 and Figure 4 which show an interaction in different languages. The preferred language and other characteristics can be dynamically varied by the user, and have an immediate effect.

As far as document presentation is concerned, every document element is assigned to a record format. Record format "full" includes record format "long", which, in turn, includes record format "short". At least one element must be assigned to the short recordFormat.

Fig. 4. Filling a queryToken in the Query Form (English language selected in User Profile)

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.).

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 issue of supporting automatic expansion of the query terms has been considered, but it seemed in contrast with the basic philosophy of the whole interface. We believe that automatic expansion can lead to high number of false hits and loss of precision. In fact, automatic term expansion could include unwanted terms, and this effect is very common in some special environments, like cultural heritage. We prefer that user will navigate on the concept tree, selecting appropriate relationships to follow and desired concepts/terms to use in formulating the query.

4.3 User Level: the Interface

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.

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 lan-
guage 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 query-Token, user can select/deselect the element as a returned element, and can ask for support from a dictionary or thesaurus. In some cases, external lists are automatically obtained invoking a web service. By default, all 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. A queryFragment is started when user selects: newFragment (so starting a new navigation upon the document-Tree). By default, queryFragments are ANDeD. A query is produced when the user specifies: buildQuery. Afterwards, 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.

4.4 Document Server Level
As already pointed out, the interface is fully independent from the search engine and document server architecture. The interface must only be aware of the supported operands and operators, just to supply adequate choices to the user. The main tasks performed at this level are: converting the incoming query into the internal search engine format, executing the query, and returning the document set.

In the current implementation we adopted, as search engine, the XCDE library [14], 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 [15] with most of the IR functionalities detailed in [16] as well other powerful string-based queries, like regular expressions and error matches.

5 Implementation Details and Case Study
As said before, at the Document Server level we have the XCDE Library. 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.

For more complete information, the reader is referred to [17]. As case study we used a little variant [18] of the document collection referred by [16]. The RDF annotation [19] of its schema [20] 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/.

6 Conclusion and Future Work

We developed a consistent framework for intelligent and controlled access to XML document collections. The document collection schema is left unchanged, but has been 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.

Finally, we want to recall that the architectural solution is fully compliant with web standards and Semantic Web environment. More precisely, the only requirement is that any document provider, who intends to make the document collection available, has to provide a XML Schema, and provide an appropriate semantic description of XML elements and attributes, as required in the Semantic Web perspective.
Acknowledgements

Authors wish acknowledge prof. Paolo Ferragina and his staff (Alessandro Perucci and Rosanna Rossi) for illuminating discussions and implementation of XCDE library. Authors wish also thank the anonymous referees, who helped in improving the quality of the paper.

References

10. The Dublin Core Home Page. URL: http://dublincore.org/
11. http://www.semanticweb.org/
17. http://www.weblab.isti.cnr.it/projects/QH/docs/