GUI for XML Documents Access using Query-By-Example Paradigm

Daniela Fonte Daniela da Cruz Pedro Rangel Henriques
Department of Informatics, 4710 Braga, Portugal
{danielafonte, danieladacruz, prh}@di.uminho.pt

Alda Lopes Gançarski
Institut Télécom, Télécom SudParis, CNRS UM Samovar
9 rue Charles Fourier, 91011 Evry, France
alda.gancarski@it-sudparis.eu

Abstract
In the context of our research project, we propose an interactive tool (GuessXQ) to perform the access to the information in a collection of eXtensible Markup Language (XML) documents. Due to the complex nature of those structured documents, the associated standard query language is also complex and, thus, not easy for most of the users. The GUI we propose does not require any knowledge about the query language, it is based on the Query-By-Example (QBE) paradigm from traditional databases. Using QBE, instead of specifying the desired components of the documents and eventual restrictions in a query, the user exemplifies those components marking them directly on a sample document picked-up from the collection. We believe this leverages the user's cognition about the object to search. GuessXQ is then responsible for the generation of the query to be treated by the information retrieval engine.

Keywords
XML, XQuery, Query-by-example.

1. Introduction

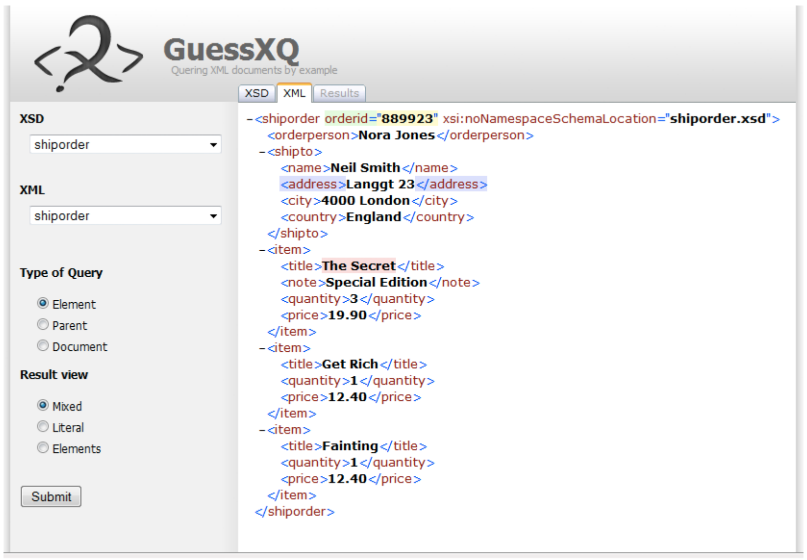
This paper addresses the problem of eXtensible Markup Language (XML) documents information access. Those documents, being structured, are accessed using specific query languages where the interesting structural components are specified, as well as restrictions over them if needed. The standard query language for XML is XQuery [BCF$^+$05]. XQuery queries are powerful but complex to write (the user must have a deep knowledge of the query language as well as the document structure). To help the user in the task of specifying his queries, some specialized editors have been developed ([Kim02], [Oxy08]), but still requiring a good knowledge level of the query language.

At the same time, thinking that "Example is always more efficacious than precept", HCI researchers proposed a Query-by-example (QBE) paradigm as a new interaction mechanism in the context of database querying [RG07]. QBE is based on the concept that the user formulates his query by filling in the appropriate skeleton tables the fields (relational projection concept) and/or restrictions on fields (relational selection concept) he intends to search for.

Due to the complex nature of XML documents querying, the QBE concept was adapted to XML retrieval [LGB07], [BC05], [NO04] by showing the XML Schema Definition (XSD) tree instead of the relational table skeleton. The system we are developing [dCFH$^+$09], called GuessXQ, also displays the XML Schema tree representation. Moreover and distinct from other approaches, the user can query the entire collection exemplifying over a sample document. Thus, elements selection and restriction is done directly in the sample document, which gives the user a clear indication of the information he is searching for.

Suppose the user is willing to search in a ship order document composed by a set of ship destinations, where each one is defined by an address, a city and a country. Consider also that his specific interest is to look for the order identifier number *889923* and get all the *addresses* where the item with the title *The Secret* whas shipped to. Adopting the QBE principle, instead of specifying the query in textual form, the user selects directly on the sample document exhibited by the QBE interface (see Fig. 1) the desired *id* (attribute value) and the elements *person* and *country*, just clicking and highlighting them.

In this paper, we fully describe the interface (actually a GUI) of GuessXQ that is responsible for all the interaction between the user and the system, providing to the user a simple but effective way of querying a collection of XML documents based on a *query-by-example* approach. Spe-



XSD: shiporder

XML: shiporder

Type of Query: [✓ Element] Parent Document

Result view: [✓ Mixed] Literal Elements

```
-<shiporder orderid="889923" xsi:noNamespaceSchemaLocation="shiporder.xsd">
   <orderperson>Nora Jones</orderperson>
  -<shipto>
     <name>Neil Smith</name>
     <address>Langgt 23</address>
     <city>4000 London</city>
     <country>England</country>
   </shipto>
  -<item>
     <title>The Secret</title>
     <note>Special Edition</note>
     <quantity>3</quantity>
     <price>19.90</price>
   </item>
  -<item>
     <title>Get Rich</title>
     <quantity>1</quantity>
     <price>12.40</price>
   </item>
  -<item>
     <title>Fainting</title>
     <quantity>1</quantity>
     <price>12.40</price>
   </item>
 </shiporder>
```

Figure 1. GuessXQ GUI (XML Tab): Sample document and Visual Query specification

cial care will be devoted to: the method used to choose the sample document from the entire collection; the visual specification of the query; the generation of an XQuery sentence from the visual specification; and the presentation of the documents retrieved.

The remainder of this paper is organized as follows. We first give an overview of the GuessXQ system discussing its architecture (Sec. 2). Sec. 3 presents the criteria used by our approach in order to select a sample document from the collection. Sec. 4 introduces the interface used to display the sample document and to allow for the visual specification of the query. This section elaborates on the way we implement the QBE paradigm. Sec. 5 describes the three different modes of querys offered by our system. Sec. 6 discusses the interface to display the information retrieved from the collection that satisfies the query. To conclude, in Sec. 7, we make some remarks and discuss the contribution of our approach, giving directions for future work.

2. GuessXQ System

After the choice of a XSD Schema (displayed in a specific window), representing the collection of documents from where the user wants to retrieve information, GuessXQ picks up a sample document from the collection and presents it (in a second window) to the user for him to specify his query. Fig. 1 shows the GUI where the user selects the components (elements or attributes) directly in the sample document. Another tab of that GUI is used to show the query inferred by the system and to display the documents retrieved.

After the *Sample Document Choice*, the document picked up is shown to the user in an interface for *Visual Query Specification*. This interface corresponds to the one presented above in Fig. 1. The *Visual Annotations* made in the previous interface are, then, translated into XQuery by a *Query Generator*. The generated query is processed by an *Information Retrieval Engine* which searches in the documents collection for the components specified in the query. The returned components are given back to the user in a *Document Viewer* interface.

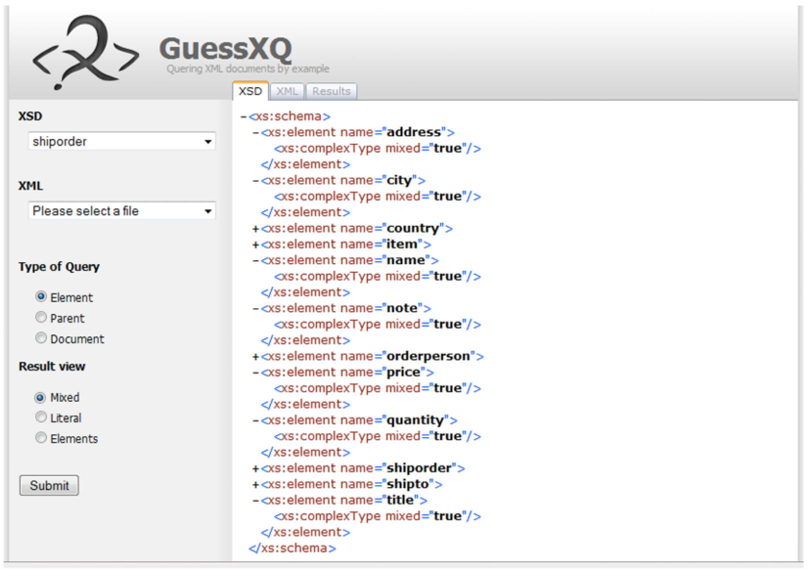
In the following paragraphs, we detail each bloc.

XML and XSD Repository The repository is a collection of XML files grouped by their schema (XSD). This Repository, in a simple way, is composed by two tables: *XMLdocs* and *XSDfamilies*. The *XMLdocs* table stores the name of the XML document, its location and its correspondent XSD family; the *XSDfamilies* table stores the name of each XSD and its location. To assess our system during development we are resorting to an archive of XML documents for testing, XAT (see [FCdC$^+$10] for details) composed of documents belonging to the following Web accessible collections: Medical Subject Headings vocabulary files (MESH), the complete plays of Shakespeare, Eurovoc - a multilingual thesaurus covering the fields in which the European communities are active, and a set of miscellaneous documents collected from different sources.

Collection Selection The collection selection allows the other modules to access the repository in a systematic and simple way. It allows the other modules to select a Schema, a collection of documents, a single document or a component of a document. It also offers an appropriate interface to allow the user to choose a document type (XSD) from the repository.

Sample Document Choice The choice of the sample document is a crucial point in our approach, since the user specifies his needs over it. Thus, there must be a well founded logic behind the selection of the sample document from the collection. We have identified several criteria which should be taken into account when choosing that document (document size, number of different elements, diversity of values, among others), as discussed in Sec. 3.

Visual Query Specification The visual query specification is done through the sample document GUI (Fig. 1), which is responsible for the interaction between the user and the



XSD: shiporder

XML: Please select a file

Type of Query: [✓ Element] Parent Document

Result view: [✓ Mixed] Literal Elements

```
-<xs:schema>
  -<xs:element name="address">
      <xs:complexType mixed="true"/>
   </xs:element>
  -<xs:element name="city">
      <xs:complexType mixed="true"/>
   </xs:element>
  +<xs:element name="country">
  +<xs:element name="item">
  -<xs:element name="name">
      <xs:complexType mixed="true"/>
   </xs:element>
  -<xs:element name="note">
      <xs:complexType mixed="true"/>
   </xs:element>
  +<xs:element name="orderperson">
  -<xs:element name="price">
      <xs:complexType mixed="true"/>
   </xs:element>
  -<xs:element name="quantity">
      <xs:complexType mixed="true"/>
   </xs:element>
  +<xs:element name="shiporder">
  +<xs:element name="shipto">
  -<xs:element name="title">
      <xs:complexType mixed="true"/>
   </xs:element>
 </xs:schema>
```

Figure 2. GuessXQ GUI (XSD Tab): the Collection Schema

system, in a way that allows the user to set the "example" for the QBE engine.

In our system, the specification of the query includes the selection of components (elements or attributes) and the possibility to restrict them to some desired value. Both of them are specified by clicking over the desired fields in the sample document. Each time a node is selected, its color is changed: this mechanism allows the user to easily see which nodes are already selected. It also uses different colors to distinguish between elements, attributes and values (restrictions) selection.

Query Generator After the query specification, the query generator module has the task of inferring the appropriate XQuery sentence. The selected nodes can be of two kinds: elements/attributes or values. An example of the former is selecting *person* element as shown in Fig. 1; the later happens for example when selecting the value from the attribute *id*. The construction of the query takes in consideration this two types of selected nodes.

Information Retrieval Engine After generating the query to retrieve document parts, a retrieval engine will access all documents in the collection where those parts appear. The engine has the capability to understand the query language and reach the interesting parts using several indexes.

Retrieved Documents Viewer The Retrieved Documents Viewer shows the results or answers produced by the retrieval engine. The user can choose, in the sample document interface (*Result view* option in Fig. 1), the type of the result to be shown: just elements; just textual values; or *Miscellaneous* (elements and text results in XML format). Fig. 4 is an example of the output visualization using *Miscellaneous* mode.

3. Sample Document Choice

In this section we will discuss the criteria used to rank documents. We identified the following metrics which should be taken into account when choosing the sample document.

1st Document size Big size files can slow down the system; but small size files can contain too little information or elements to aid the user selection. This metric can be used as a delimiter to complement the other ones; the principle is to not choose a file bigger than a predefined size.

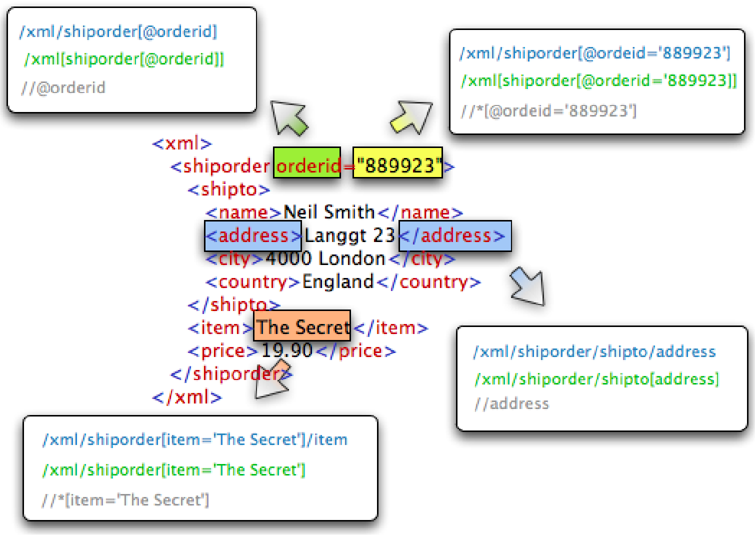
2nd Number of components It is important to take into account the number of components in the sample. On one hand, if the file has too many components, it can be too cluttered for the user to select the desired example. On the other hand, if the document has few components, maybe it does not contain all those the user needs.

3th Number of different components To counteract some of the shortcoming of the previous metric, it may be interesting to look at the number of different components in a file. This way, if a file contains almost all the elements and attributes present in the schema, the user gets a more complete variety of elements to specify his needs.

4th Diversity of Values As stated before, the main innovation of our QBE approach is the capacity of the user to see sample data and not just the structure (schema) of the queried documents. Therefore, a metric guaranteeing the diversity of data is important. Having different values for the same component allows the user to better understand the fields in the document he is querying. However, similar to the other metrics, if there is too much diversity, the example document may become too big.

5th Number of commonly used components and values It can be interesting to have a sample document as similar as possible to the majority of the documents in the collection. This means that the sample should contain commonly used components and values which may be the most interesting for the user example. A component/value is considered to be common if it is in the top N more frequent components/values of the collection.

As seen, each criterion has its own merits and shortcomings, so they must be used together in a meaningful way.



```
/xml/shiporder[@orderid]
/xml[shiporder[@orderid]]
//@orderid
```

```
/xml/shiporder[@ordeid='889923']
/xml[shiporder[@orderid='889923']]
//*[@ordeid='889923']
```

```
<xml>
  <shiporder orderid="889923">
    <shipto>
      <name>Neil Smith</name>
      <address>Langgt 23</address>
      <city>4000 London</city>
      <country>England</country>
    </shipto>
    <item>The Secret</item>
    <price>19.90</price>
  </shiporder>
</xml>
```

```
/xml/shiporder/shipto/address
/xml/shiporder/shipto[address]
//address
```

```
/xml/shiporder[item='The Secret']/item
/xml/shiporder[item='The Secret']
//*[item='The Secret']
```

Figure 3. Query Mode: the different XPath expressions associated to each component.

4. Visual Query Specification

GuessXQ offers a simple and intuitive GUI that allows the visual specification of the query, without the need of an advanced knowledge of the XQuery language. In this section, we describe the interaction between the system and the user through this interface (shown in Fig. 1, as referred in Sec. 2), to explain the QBE concept proposed.

As previously said, the user starts by selecting the intended collection choosing a Schema and the system shows its content in the XSD Tab, as illustrated in Fig. 2.

After the Schema selection GuessXQ suggests a sample document, as fully described in Sec. 3. To select the desired components (elements or attributes), the user can click directly on the specific item of the document (exhibited in the XML Tab, as illustrated in Fig. 1) and restricts their value. To improve this interaction, each time a component is selected its color is changed. This enhancement is obtained using different colors, according to their types: elements are highlighted in blue; attributes in green; PCDATA content is highlighted in red; and the attribute values in yellow. To unselect an highlighted component, the user just has to click again over it.

To improve the perception of the restriction implied by each selected item (in terms of the final query), when the user puts the mouse cursor over it, GuessXQ shows a tip with the correspondent XPath expression.

To simplify the search of the components to select, the system provides a feature to expand or retract blocks inside the document, more specifically elements with children nodes. Fig. 2 illustrates this feature: by clicking on "-" (*minus*) sign, the correspondent block retracts, and only its element name is displayed, preceded now by a "+"(*plus*) sign. To expand it again, the user just clicks on the "+" sign, and the entire element subtree will be displayed.

After finish the visual query specification, the user must submit it so that GuessXQ starts the IR process, applying to the entire collection this query (see Sec. 6 for details).

5. Query Generator

GuessXQ generates the XQuery query based on the union of the paths obtained for each selected component. These paths are assigned to each document node according to three different modes of querying: *element-oriented*, *parent-oriented* and *document-oriented*, which user can choose in the *Type of view* option (as illustrated in Fig. 4). By default, system selects the *element-oriented* mode.

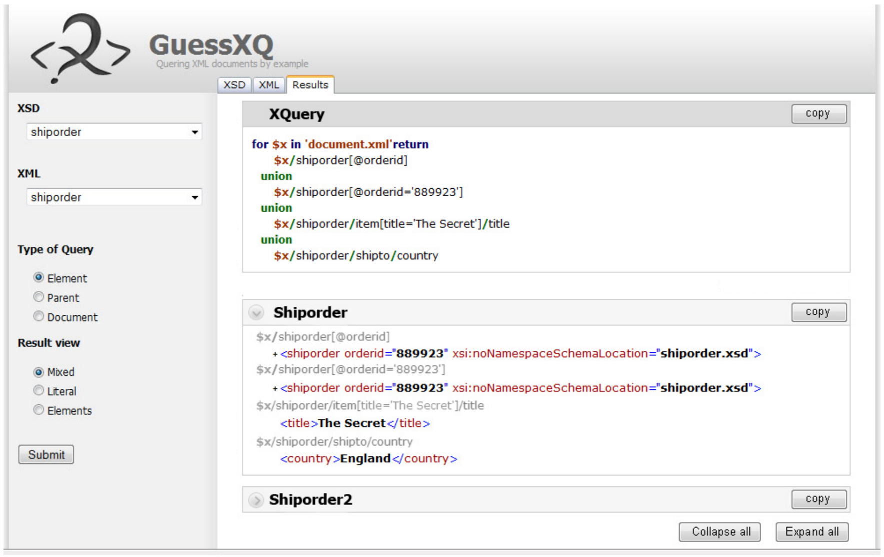
In the *element-oriented* mode, the path reflects the correspondent *absolute* location path from the root node to the selected component (as illustrated in Fig. 3 by the first path shown in each selection). In the *parent-oriented* mode, the generator assigns to each document node the *absolute* path from the root node to its parent (as illustrated in Fig. 3 by the second path in each selection). In the *document-oriented* mode, the generated XPath expression allows the search of the selected component on the entire document, independently of its position on the document tree (as illustrated in Fig.3 by the third path in each selection).

Before the submission, the user must select the type of view he wishes for the retrieved answer, as will be described in Sec. 6.

6. Retrieved Document Viewer

Concerning the visualization of the answer retrieved by the search engine, the user can choose one out of three modes (as said in Sec. 5): only the correspondent node names (*elements*); only the *literal* (PCDATA) values; or a *mixed* of both *elements* and *literal*, which shows the entire XML nodes resulting from the search. By default, the system outputs the results in *mixed* mode.

After the submission of the desired selection, GuessXQ shows in the Results Tab (as depicted in Fig. 4) the respective query and the result of applying it to the entire collection of documents (actually the desired output). First, the system displays the result extracted from the sample document, followed by the results (non-empty) extracted from each other document in the collection. Each result



```
for $x in 'document.xml'return
    $x/shiporder[@orderid]
  union
    $x/shiporder[@orderid='889923']
  union
    $x/shiporder/item[title='The Secret']/title
  union
    $x/shiporder/shipto/country
```

```
$x/shiporder[@orderid]
  +<shiporder orderid="889923" xsi:noNamespaceSchemaLocation="shiporder.xsd">
$x/shiporder[@orderid='889923']
  +<shiporder orderid="889923" xsi:noNamespaceSchemaLocation="shiporder.xsd">
$x/shiporder/item[title='The Secret']/title
    <title>The Secret</title>
$x/shiporder/shipto/country
    <country>England</country>
```

Figure 4. GuessXQ GUI (Results Tab): Retrieved Document Viewer

is loaded into a selector, to improve the usability: the user can expand the selector by clicking over it, or retracting it again by doing the same action (as illustrated in Fig. 4).

The system also allows to expand/collapse all the selectors with one click and offers the chance to copy the *query* or each particular output to the clipboard. In this way, we aim at offering a versatile and user-friendly interface.

7. Conclusion

This article presents new contributions for XML query specification in a user friendly interface. As far as we know, no other previous work addresses the framework of QBE in the context of *XML information access based on a sample document from the collection*.

In this paper, we focus on the general interface provided by our tool to support the visual specification of a query based on a example pointed out over a sample document. This interface also exhibits the query inferred from the selection, applies it to the entire collection, and at last displays the retrieved answer (a list of documents or theirs parts).

As future work, we plan to implement different methods to choose the sample document, as well as to finish other aspects under construction by now (like output visualization). After this, we think about making experiments for tuning the score computation parameters. As soon as possible, we intended to step forward to a second experimentation phase; GuessXQ interface will be assessed, testing the system with real users to measure their level of satisfiability.

References

[BC05] Daniele Braga and Alessandro Campi. Xqbe: A graphical environment to query xml data. *World Wide Web*, 8(3):287–316, 2005.

[BCF+05] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, and J. Simeon. Xquery 1.0: An xml query language. w3c working draft. http://www.w3c.org/TR/xquery, 2005.

[dCFH+09] Daniela da Cruz, Flávio Xavier Ferreira, Pedro Rangel Henriques, Alda Lopes Gançarski, and Bruno Defude. Guessxq, an inference web-engine for querying xml documents. In *INForum'09 — Simpósio de Informática*, pages 322 – 325, Lisboa, Portugal, September 2009. Faculdade de Ciências da Universidade de Lisboa.

[FCdC+10] Daniela Fonte, Pedro Carvalho, Daniela da Cruz, Alda Lopes Gançarski, and Pedro Rangel Henriques. Xml archive for testing: a benchmark for guessxq. In *XATA 2010 — XML, Associated Technologies and Applications*, Vila do Conde, Portugal, May 2010.

[Kim02] Larry Kim. *The XMLSPY Handbook*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[LGB07] X. Li1, J. H. Gennari1, and J. F. Brinkley. Xgi: A graphical interface for xquery creation. In *Proceedings of the American Medical Informatics Association Anual Symposium*, pages 453–457. American Medical Informatics Association, 2007.

[NO04] Scott Newman and Z. Meral Ozsoyoglu. A tree-structured query interface for querying semi-structured data. *Scientific and Statistical Database Management, International Conference on*, 0:127, 2004.

[Oxy08] Oxygen xml editor. http//www.oxygenxml.com, 2008.

[RG07] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*, chapter 6 - Query-by-Example (QBE). 2007.