International Journal of New Computer Architectures and their Applications (IJNCAA) 7(2): 50-55
The Society of Digital Information and Wireless Communications, 2017 ISSN 2220-9085 (Online); ISSN 2412-3587 (Print)

Querying XML and Relational Data

Hassana NASSIRI[1, a] Mustapha MACHKOUR[2, a] Mohamed HACHIMI[3, b]
[a] Laboratory of the Computing Systems and Vision
[b] Laboratory of Engineering Sciences
University Ibn Zohr, Agadir, Morocco
[1] hassana.nassiri@edu.uiz.ac.ma
[2] machkour@hotmail.com
[3] m.hachimi@uiz.ac.ma

ABSTRACT

There has been a growing need for querying heterogeneous data sources, namely XML and Relational databases. Since the relational model is the most data model used to manage data for years. Similarly, the eXtensible Markup Language (XML) is quickly emerging as the de facto standard for data exchange over the Internet. Hence, bridging these two models is surely need. Furthermore, each database system uses a particular query language to manipulate data. So, users need to know each query language of each data model. To this point, we aim to define a system to retrieve data regardless of the nature of the model used and eliminates the burden of learning new languages. In such way, the existing users' knowledge about a query language will be enough and will meet the purpose. Thus, this paper addresses the problem of accessing both XML and relational data, by using a unique query language expressed with whether SQL or XPath. We rely on a new approach in the translation process to convert the user query into the suitable query language according to the nature of the data interrogated.

KEYWORDS

SQL; XPath; Data Model; XML; Relational Database; Translation; Mapping

1 INTRODUCTION

We choose to discuss the eXtensible Markup Language (XML) and relational data model and make them the first models under study in this contribution, as they are increasingly complementary and related in practice. The large volume of the research introduced to integrate these two models is another proof that both relational and XML databases should exist. The Relational Model has long been a successful, dominated and popular Data model, and still widely used by most organizations to manage data. Besides, the Structured Query Language (SQL) is the most popular, basic and the standard query language for managing and querying data. SQL is user-friendly, and almost everything we need to do in manipulating a database can be fulfilled using SQL. Similarly, in the last years, XML is quickly emerging as the de facto standard for data exchange over the Internet. So, bridging these two models is necessary.

Each database system uses a particular query language to manipulate data. So, to query a database, users need to adapt the query language on it, for instance, use SQL to extract data from relational and XPath, for example, to get data from XML. But most of the time, it is a hard task for users to learn new query languages and to control the use of all of them. To this point, we aim to define a system to extract data regardless of the nature of their model. By our system, the existing users' knowledge about a query language will be enough to get what they want. They can use SQL as they can use XPath to retrieve data from both XML and relational data model.

The rest of the paper is organized as follows: Section 2 provides some related works. Section 3 discusses the objectives of the system and explains the translation process to convert queries. Finally, section 4 draws the conclusion.

2 RELATED WORK

There has been considerable interest related to XML and relational database. We focus on those approaches that have proposed a query translation tool to solve the problem of querying different databases. From systems studied in the literature, we have noticed two ways to handle the problem and met our purpose: (1) translate SQL to an XML query language XPath or XQuery to query XML database (2) the opposite, translates an XML query language to SQL. We will divide them into two parts: from relational to XML [1] [2] [3] [4] [5] [6] and from XML to relational [7] [8] [9] [10] [11] [12] [13]. In the rest of this section, we will give a brief description of each research aforementioned.

[1] discusses a way to query XML documents using relational database system by designing an SQL Interface for XML databases that can convert SQL queries to XPath expressions and extract data from XML documents.

[2] proposes a framework which permits users to access XML databases using SQL. It is an automatic converter of the user's SQL join (Left, Right, and Full) queries into XPath expressions and describes the detailed steps of SQL to XPath conversion along with their algorithms.

[3] examines a way to modify XML data using SQL (INSERT and UPDATE) by proposing a framework to translate these SQL queries to XUpdate expressions, and presents the algorithms to do so. Users can manipulate the XML data through the query language SQL or XUpdate.

[4] introduces a framework to transform SQL statements to XPath expressions, so that accessing XML and relational database can be done using SQL.

[5] proposes a framework that can transform easily SQL statements to XQuery expressions. So users can access both XML and relational database through SQL using their framework.

[6] the ROX explains an approach to translate SQL to XQuery and to query XML data stored in its native format, through SQL interfaces.

[7] uses Relational Database Management System (RDBMS) to store and retrieve XML documents by translating them into tables in RDBMS and storing them in a shredded schema. The user can use XQuery query language, which is translated into SQL Queries. Then, the returned result will be XML documents.

[8] discusses algorithms to translate queries from XPath to SQL. And presents a Bi-LAbeling based System: BLAS [9] for Processing XPath queries over XML data.

[10] proposes an approach to translating a practical class of XPath queries over (recursive) DTDs to SQL queries. Also, it discusses algorithms for rewriting an XPath query over a recursive DTD.

[11] presents a translation of XQuery expressions from a comprehensive subset of XQuery into a single SQL expression.

[12] Presents Agora data integration system [13], a way to integrate relational and tree-structured data sources, in particular XML, under an XML global schema, and to translate queries from XQuery to SQL.

The novelty of our work lies in its ability to retrieve data regardless of the nature of the models with any query language of these models. That means that users can use XPath or SQL to retrieve data from XML or/ and relational databases. In fact, making one query sufficient to retrieve data from heterogeneous databases is among our goals, since users and even developers may not be familiar with many query languages at a time. Using our system, whatever the query posed users can extract data stored in different databases.

3 PROPOSED SYSTEM

As we know, there is a correspondence between the data model and its query language. For instance, we use SQL to extract data from relational and XPath, for example, to get data from XML. That means that users need to know each query language of each data model. Mostly it is a difficult task for users to support all of these query languages because each language has a defined syntax, particular specification and probably difficult to learn.

The principal objective of our system is to define a way to query XML and Relational data sources using one single query expressed with either SQL or XPath as explained in Figure 1. And figure out an efficient method to translate the user query into the suitable query language according to the nature of data interrogated. We think that it is more convenient to build an intermediate format to pass between phases rather than repeat the whole procedure for each query language and break down the translations process into multiple steps. Each one has a particular task to accomplish. The architecture of our system in a whole is detailed in Figure 2, with all the components, which they are in turn explained in the following sections.

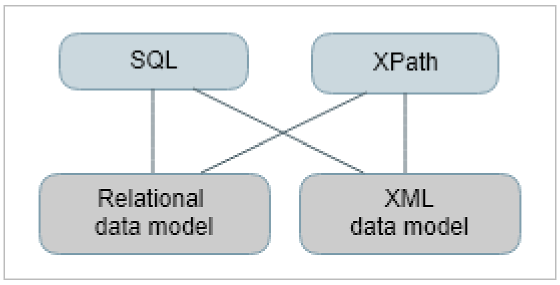

Figure 1. [14] One query to retrieve data from XML and/or relational data models

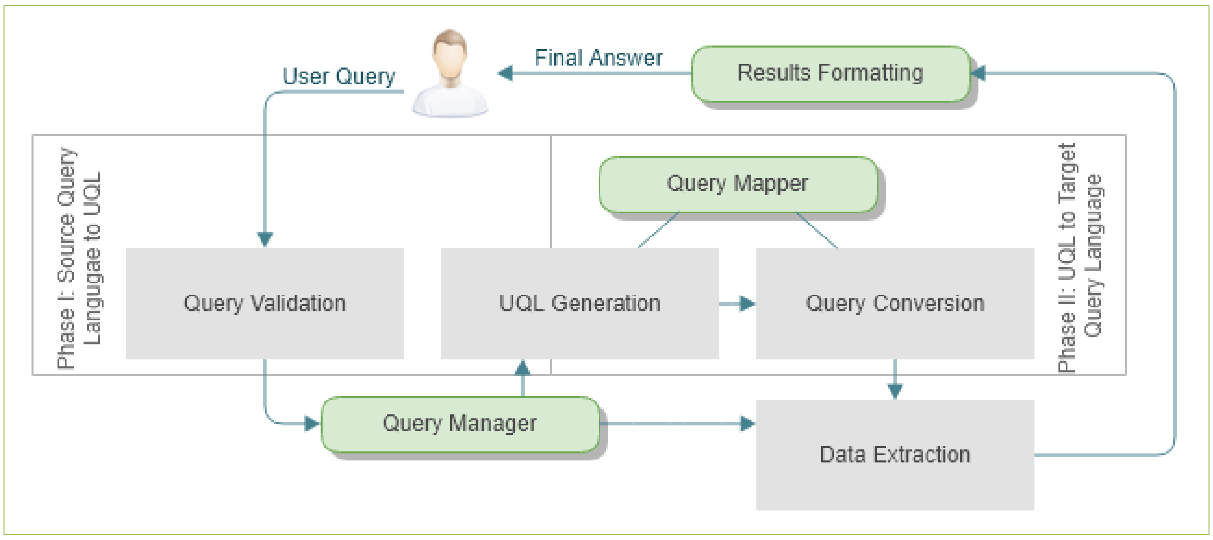

Figure 2. Architecture of the overall system

3.1 Intermediate Representation

We inspired from the compilation notion and adopted the same principle in our approach to translating queries. The system reads the Source Query Language, then translated into an Intermediate Query Language(IQL) which is then translated to its Target Query Language as shown in Figure 3.

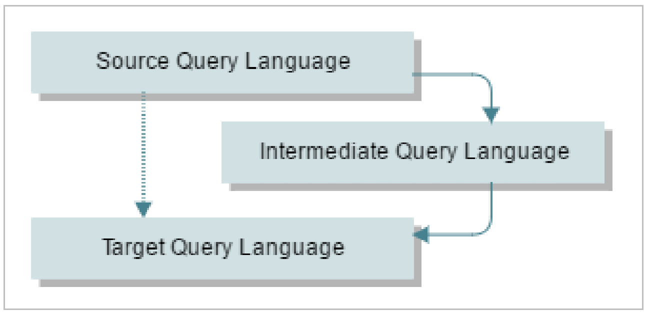

Figure 3. The principal of the translation

Why at all we need an IQL in the translation process, instead of simply translate the query directly into its target and skip the intermediate presentation phase? In fact, if we translate the source language into its target language without generating intermediate language, then for each new language, a full native translation is needed. Using intermediate code eliminates this need by keeping a portion same for all. Then the second part is changed according to the target language. Thus, it becomes easier to add more languages in our system.

The benefits from using an IQL are multiple, we cite for example the fact that it is more general and capable of representing different Query Languages; can operate on the lowest level semantics; can increase possibilities to use more independent transformations and optimizations; can be an aid to switch between several query languages, and conversion between two languages will be through it, also it enables the system to be broken up into multiple components, more manageable and simpler, thus benefiting from modularity.

3.2 Universal Query Language

In our system, we refer to IQL by the universal query language, the representation of a query between the source and target query languages. We know that a good intermediate representation is one that is independent of the source and target languages so that it maximizes its ability to be used in different cases. For that, XML was our candidate to represent our IQL. XML allows separating the contents of the presentation. This makes it possible, for example, to display the same document on different applications or devices without necessarily creating as many versions of the document as we need of representations. Also, it has many qualities such as readability, universality, portability, integrability, and extensibility.

3.3 How it Works?

In this section, we will explain the operation of the system, which consists of four phases as shown in Figure 4: (1) query validation, (2) UQL generation (3) Query conversion and (4) Data extraction.

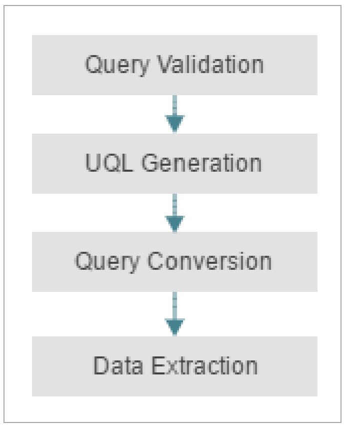

Figure 4. The system's phases

The procedure begins with the user query as expressed in Figure. 5, then verified in the query generation phase, if it is valid we continue the rest of the process, else we ask the user to enter a valid query, this is repeated until getting a valid query. After that comes the query checker role, to decide if the query really should be translated. If that the case we continue the translation process and use the query mapper module to map between each part of the query with the correspondent one in our UQL, else we skip it and pass the query directly to the query executor, then extract data and finally format the result and return the final answer to the user.

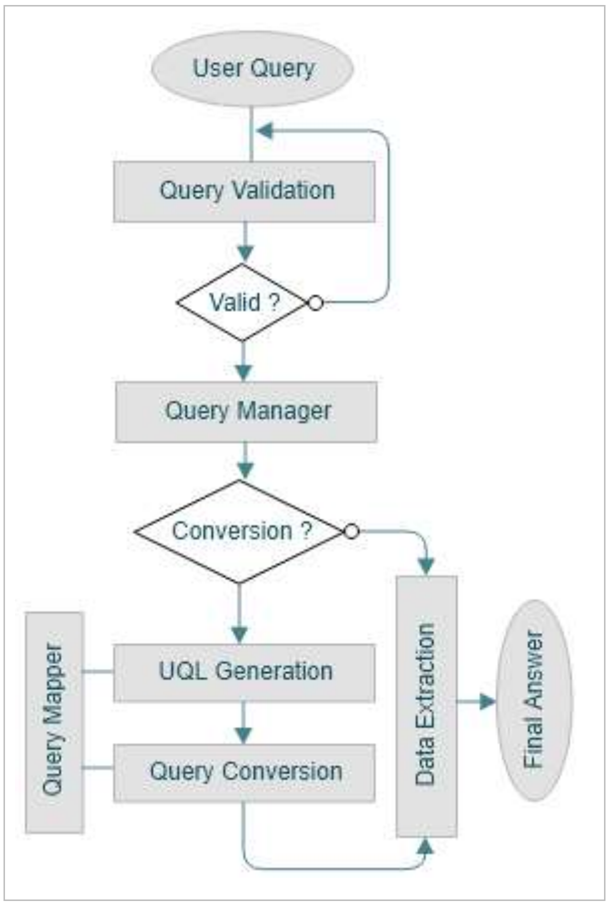

Figure 5. How it works

3.4 Translation process

In this section, we present how to convert queries from a language to another. As shown in Figure. 6 the translator suggested herein maps each input of a query language to an output. To do so the procedure is broken down into multiple phases, and each phase has a defined task. The translation process includes two parts: I. from the source query language to UQL, II. From UQL to target query language. This happens through the first three phases of the system: (1) Query Validation (2) UQL Generation and (3) Query Conversion.

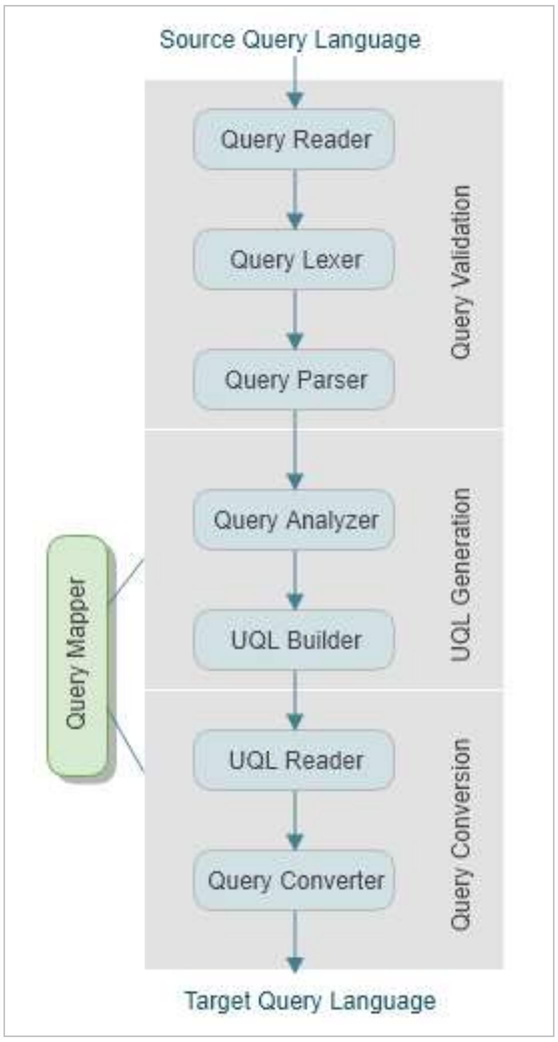

Figure 6. The translation process

3.4.1 Query validation

The Query validation starts with syntax verification of the input given by the user which is analyzed then to extract its portions to appropriate variables. The semantic information's related to the query are captured here.

- Query Reader: provides a uniform interface between users and system, and read their queries.
- Query Lexer: this is the lexical analysis or scanning phase, the lexer converts the groups of characters of the query written by the user into tokens.
- Query Parser: the query parser converts the groups of tokens into a parse tree, a data structure that reflects the syntactic structure of the input query.

3.4.2 UQL generation

The UQL Generation phase is responsible for building the UQL via the information received from the query validation phase. It has two components:

- Query Analyzer: In this step, we split the information extracted and taken it into variables which will be used to proceed further operations.
- UQL Builder: After getting all the needed variables from the query analyzer, it becomes easier to form the UQL by filling each part of it with the suitable information retrieved from these variables using the Query Mapper.

3.4.3 Query Conversion

The Query Conversion phase where actually the translation takes place by the query converter with the aid of the query mapper. Again, it consists of two components.

- UQL Reader: recalls the query mapper again to prepare the UQL parts that should be mapped with the target query language.
- Query Converter: Now that we have our intermediate format, all what we need to do is convert it into the target query language which is chosen according to the nature of data source.

3.5 Query Manager

Not much work is involved in to cases: (i) if the user uses SQL to query relational databases (ii) if he or she uses XPath to query XML database. In other cases, the translation is proceeded. This kind of decision is made by the query checker, it is the one that make sure if we really need to enter the translation process or skip it and directly execute the query and extract data.

3.6 Query Mapper

Contains the steps to link each part of a query to the suitable part of the UQL, and vice versa.

4. CONCLUSION & FUTURE WORK

We try to eliminate the burden of learning new languages so that querying each data model with the correspondent query language is not a problem anymore; users can use whether SQL or XPath to query both XML and relational data. We rely on a new approach in our translation process to convert the user query using an intermediate query language as an intermediate format to pass between the source query language and its target according to the nature of the data interrogated.

XML and relational models were the first models to discuss in our research because of all the attention they received in data integration filed since they have some powerful tools and many merits in either data exchange and data management and extraction. So, until now, only XML and relational data model are supported in our system, but in future work, we aim to enhance the features, and the merits of the system proposed, to integrate other data models and support other query languages.

REFERENCES

1. H. A. Kore, S. D. Hivarkar, N. K. Pathak, R. S. Bakle, and P. S. S. Kaushik, “Querying XML Documents by Using Relational Database System,” vol. 3, no. 3, pp. 5322–5324, 2014.
2. K. Bhargavi and H. S. Chaithra, “Join queries translation from SQL to XPath,” 2013 IEEE Int. Conf. Emerg. Trends Comput. Commun. Nanotechnology, ICE-CCN 2013, no. Iceccn, pp. 346–349, 2013.
3. M. Vidhyap and P. Samuel, “Insert Queries in XML Database,” pp. 9–13, 2010.
4. P. M. Vidhya and P. Samuel, “Query translation from SQL to XPath,” 2009 World Congr. Nat. Biol. Inspired Comput. NABIC 2009 - Proc., pp. 1749–1752, 2009.
5. S. Jigyasu et al., “SQL to XQuery translation in the aquaLogic data services platform,” Proc. - Int. Conf. Data Eng., vol. 2006, p. 97, 2006.
6. A. Halverson, V. Josifovski, G. Lohman, H. Pirahesh, and M. Mörschel, “ROX : Relational Over XML,” Proc. 30th Int. Conf. Very Large Data Bases, pp. 264–275, 2004.
7. Y. Bin Chiu, H. H. Chen, C. Y. Liu, S. C. Chen, and C. W. Hung, “Efficient Storage and Retrieval of XML Documents Using XQuery,” Adv. Mater. Res., vol. 779–780, pp. 1685–1688, Sep. 2013.
8. Y. Chen, S. B. Davidson, and Y. Zheng, “A bi-labeling based XPath processing system,” Inf. Syst., vol. 35, no. 2, pp. 170–185, 2010.
9. Y. Chen, S. B. Davidson, and Y. Zheng, “BLAS: An Efficient XPath Processing System,” Proc. ACM SIGMOD Int. Conf. Manag. data, pp. 47–58, 2004.
10. W. Fan, J. X. Yu, J. Li, B. Ding, and L. Qin, “Query translation from XPath to SQL in the presence of recursive DTDs,” VLDB J., vol. 18, no. 4, pp. 857–883, 2009.
11. D. DeHaan, D. Toman, M. P. Consens, and M. T. Özsu, “A comprehensive XQuery to SQL translation using dynamic interval encoding,” Proc. SIGMOD 2003, pp. 623–634, 2003.
12. I. Manolescu, D. Florescu, and D. Kossmann, “Answering XML Queries on Heterogeneous Data Sources.,” Vldb, vol. 1, pp. 241–250, 2001.
13. I. Manolescu, D. Florescu, D. Kossmann, F. Xhumari, and D. Olteanu, “Agora: Living with XML and relational,” Vldb, pp. 623–626, 2000.
14. H. Nassiri, M. Machkour and M. Hachimi, "Integrating XML and Relational Data", Procedia Computer Science (2017) pp. 422-427.