Flexible Queries in Distributed Structured and Semistructured Documents

Carsten Lecon
Fachhochschule Lübeck, lecon@fh-luebeck.de

Abstract

The goal of the thesis is to solve the task of defining a flexible query language for distributed structured and semistructured data and to implement this language. For example, data can be represented as HTML files in the file system, whereby an XML document contains metadata about these files. Hence, these two data sources have to be merged appropriately. In order to satisfy the needs of the heterogeneous group of users, a flexible adaptable query language is required. Therefore a query model is developed which serves as a basis for an abstract syntax for query languages; this abstract syntax can be translated in concrete textual as well as graphical query languages. An implementation of the ideas proposed in the thesis is done in the area of virtual courses, especially in the project "Virtual University of Applied Sciences".

1 Introduction

In many areas there exist distributed structured and semistructured data ([7]). For example, data can be represented as HTML files in the file system, whereby an XML document contains metadata about these files. Hence, these two data sources have to be merged appropriately. The problem of defining an adequate query language for these data is an actual task. However, especially in the area of virtual courses there are many requirements which can not be solved easily by existing solutions. In order to satisfy the needs of the heterogeneous group of users, we develop a model for a flexible adaptable query language called *Quings*. This query model serves as a basis for an abstract syntax for query languages; the abstract syntax can be translated in concrete textual as well as graphical query languages. Most of these ideas are implemented in the area of virtual courses, especially at the project "Virtual University of Applied Sciences" ([1]).

The thesis touches above all data models, query languages and integration of heterogeneous information sources.

The paper is structured as follows: First, the research methodology is depicted (section 2). Next, we will list the problems which arise in the application of virtual courses and which we want to solve (section 3). We show our approach to solve these problems comparing to related works (section 4). In section 5 we describe the current state of the thesis. The paper concludes with an overview over the next tasks (section 6).

2 Research methodology

The steps how the work is done are depicted in Fig. 1.

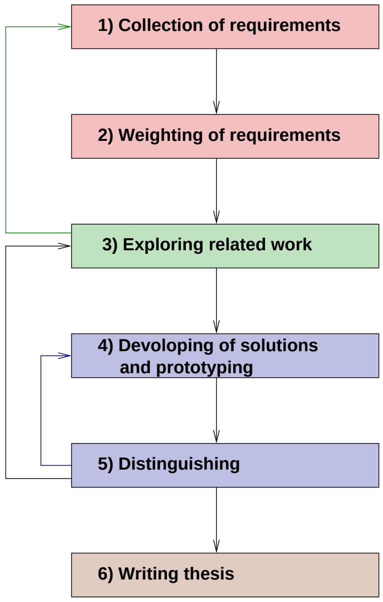

Figure 1: The working process

After the collection of the requirements given by the application ”virtual course”, these requirements are weighted. Since not all requirements can be treated (time limitation), only the most important aspects have to be chosen.
Then we look at works which are done in the particular area. A new solution is proposed which is appropriate for our application. This is the main and most important part of our work. After building the theoretical background, these ideas are evaluated by implementing a prototype.
The steps described above form an iterative process. Momentarily an iteration of the steps three to five takes place. A data model and an abstract query language are specified and implemented; the query interface is a HTML form (see section 5).

3 The problem

3.1 Description of the problem

We will describe the problem which arises in the search function in virtual courses[1]. Some aspects described in the following are well known requirements of query languages.
The problems fall into three categories:

- The query language.

[1] One can easily see that this problem arises in other contexts as well.

- The underlying data model(s).
- The software system in which the query language is implemented.

We will look at every aspect separately beginning with the data model.

3.1.1 Data model

The virtual courses of a virtual university are realized by a heterogeneous group. Therefore, they are often represented in a different way. For example different metadata is used to describe the content.
From the student's point of view the subject taught consists of pages (mostly HTML pages) which contain sections and/or multimedia objects. In order to support user adapted learning it is desirable to generate pages on demand; this means that the data needs to have appropriate metadata and the pages have to be decomposable into parts, if wanted.

3.1.2 Query Language

Often the data is not stored in a database management system (flat files in a file system, for example HTML or text files, and WWW documents). This means, that a search is not supported on the data source's part.
The specification of the search functionality is done by the developers of virtual courses, they define a kind of *view*. Because of the heterogeneity of the developers of virtual courses and their different knowledge background it is necessary that this specification should be quite simple to define.
The query language does not need to provide data manipulation operations because a once generated virtual course needs not to be – actually must not be– modified by the users. Furthermore the query language does not need to support methods.
The data of a virtual course can be distributed over multiple data sources like files (HTML-pages), a relational database system (corresponding metadata) and so on. This has to be taken into consideration when developing a query language.
The query language has to offer path expressions and recursive queries, for example to determine the prerequisites of a subject.
Because of the heterogeneity of the users – normally students –, the query language has to be adapted to the special needs of the students. This concerns the provided search functionality as well as the used syntax. It is desirable to offer textual languages, for example an SQL like language, and graphical languages, for example HTML forms, to the user.
The results of a query should be sorted by relevance. Furthermore, to allow a *query refinement*, the results of a query have to be reusable in order to serve as a basis for a new search. A query itself has to be stored for future use.
It has to be considered that the data may be distributed over several data sources at the same time. Possibly, different data sources point to the same information; for example the course data are represented as flat HTML files whereas the corresponding metadata can be found in an XML document or a relational database. Therefore data integration must be provided.

Temporal queries are useful, for example to look for modified contents.
Normally, the user does not need to know the schema of the data. However, sometimes it can be useful to provide schema information to allow more specific queries.
The query functionality can be added to existing virtual courses later (after developing of the content of the course). Hence, the query language should be "light-weighted", i.e., it should use little memory, search functions out of the ordinary should be ignored and it has to be easily adaptable to different query syntaxes.

3.1.3 Implementation

Although there exists the *flat rate* for Internet access, studying in the Internet is often expensive, so that students learn offline as well – mostly at home. Thus, the system must work online and offline.
Furthermore, it is important to guarantee an efficient query processing.

3.2 Main aspects which will be investigated in the thesis

We will mainly concentrate on these aspects (sorted by estimated volume):

- An abstract query language for searching structured and semistructured data.
- Examples of concrete query languages based on the abstract query language on top.
- Efficient query processing based on the abstract query language above.
- The underlying general data model.
- Implementation of these concepts in the area of virtual courses.
- Evaluation in the area of virtual courses.

4 How to solve the requirements

4.1 Queries in virtual courses

We examine the requirements of the section above and will show how these can be solved in consideration of existing works.
We look at queries in virtual courses. Some typical kinds of queries are:

- Full-text search: Searching the content of virtual courses, using regular expressions ideally.
- Searching metadata: Examples of metadata are title, level, content type (definition, example, exercise, etc.).
- Structure search (hierarchical): For example looking for all sections of a chapter having a specific title; looking for the third page beginning from the actual page (path expression).

- Structure search (net topology): Searching of links, for example looking for all pages to which other pages refer; looking for all pages within a specific learning trail ([16]).

4.2 Data model

The problem is that the same kind of data can be represented differently. For example the following representations of links are possible:

- Relational database: Foreign keys or other attributes.
- Object oriented database: Object references.
- XML document(s): `IDREF(S)` attributes or even `XLink` or `XPointer`.
- HTML file(s): `<A HREF ...>` tags.

Therefore we will transform these different kinds of views of the same data to a simple uniform form: We use an attribute *link* (this name is arbitrarily) to symbolize the reference of two (or more) objects (in the sense of virtual courses: pages). This attribute is assigned to all objects (pages) which have the source of a link, the domain of this attribute is object-valued. In this way, structural information is converted to structureless information. This makes the search easier because only attribute values have to be considered when processing a query. In general, the data is distributed over three sets: A set of objects, a set of attributes which are assigned to objects and a set of values representing attribute values of objects. We call this representation of data *Quing Data Model* (*QDM*; *Quings* are explained later in this section). The *schema* in the Quing Data Model is expressed by the semantics of the attributes, that is, for example the attribute ”title” stands for the title of a page in a virtual course; the same attribute can be used in another application as something which one will be addressed by (for example ”Professor.”). Using this data model, references between objects are not part of the objects itself. Thus an external representation of links takes place, so that broken links can easily be detected or avoided, respectively.
We could transform the data into another overall data model like *OEM* ([5]), which is the de facto standard semistructured data model ([4]), the ODMG data model ([2]) or the data model of the *Garlic* project ([15], [12]). These models are bound to a specific implementation of an information system: *Lore*, an object oriented database system or the Garlic system. To facilitate online as well as offline search the specific data management system has to be installed at the student's home, which is a not acceptable requirement for the students of virtual courses. The data has to be stored as files to achieve offline access as well as online access. In order to achieve an efficient access to data, connected data should not be spread out over multiple files. For example, if an object oriented modeling of data is chosen, it is disadvantegous to represent each class as one file. When searching pages of virtual courses, it is desirable to choose as few files as possible, at best one file only. This has influence on the efficiency of query processing.

Furthermore, the data models mentioned above are too complex: Mostly, objects are not instances of (known) classes, the concepts inheritance and abstract data types as well as methods (operations) are not used. Using *ODL* (ODMG data model), OEM syntax or *GDL* (Garlic project, [15]) the specification of the data would be unnecessarily complex because only a part of it would find a use.
In the area of virtual courses we have the advantage that we need only reading access to the data. Therefore we do not need a data management system. Of course, the data can be stored in a database management system as well, but this works only if being online. With the separation of the query functionality and the query processing in our implementation (see below, especially Fig. 2 and Fig. 3), the use of the data can be different being online and offline, respectively.

4.3 Implementation of queries

The components of a query represent a search functionality (and at the same time a view of the data) and form the "ingredients" of a query. That is the reason we call these components *Query Ingredients* (*Quings*).
The data for a query can be distributed over multiple information sources. Hence, the query processing is done using a standard architecture for integrating heterogeneous information sources (Fig. 2). The information is distributed over multiple data sources. So-called wrapper transform the data of the data sources into our data model. This corresponds to the data transformation at the wrapper in the Garlic project ([12]). The metadata of the different data sources may overlap or the semantics of same attributes are different; furthermore it should be possible to extend a data source by additional metadata (if annotations to some data are necessary and there is no writing-access to the data). Hence, a data integration and schema merging is necessary. In the Garlic project, this corresponds to the data transformation in the the middle-ware.
We introduce a further abstraction layer. For the *Query Ingredients* (*Quings*; see Fig. 2 and Fig. 3) no schema information and no information about the data formats is necessary. Only the meaning of the search functionality of the Quings is required, in a colloquially manner. The assignment to a concrete query processing is done at runtime, dependent of the actual circumstances (for example online and offline access to the data of a virtual course). The translation of this colloquially described search into a real executable query is done by assigning a Quing to a *search view* (Fig. 2 and Fig. 3), this search view corresponds to a particular data schema. In turn, the schema can be put together by other schemata (data integration, see Fig. 2 and the association "contains" in Fig. 3). This schema information results from wrapping data from (external) data sources. The search view is responsible for translating the desired query to an adequate access to the data.
The data can be transient or materialized. In the latter case a storage mechanism has to be provided to store the data in the format of the Quing Data Model. A relational database system is well suitable. The queries can be formulated in SQL. However, this alternative only works being online.
The additional abstraction layer (Quing) is of advantage: Search functionalities or whole queries can be specified independent of a particular information source and

even independent of a particular schema. The semantics of Quings – represented by the search views – are dependent on the use of the attributes of the data (represented in the QDM). Thus, not only a data management system can be exchanged, but also the semantics of the attributes of the data can vary. This is useful if a more suitable representation of the data can be found or even a completely different (general) data model is used.

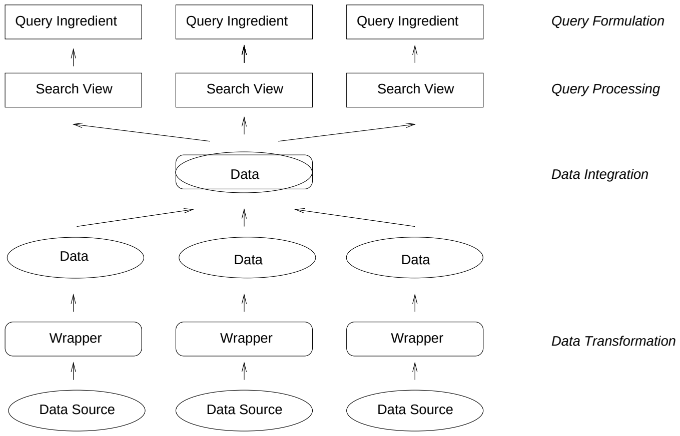

Figure 2: Quing architecture

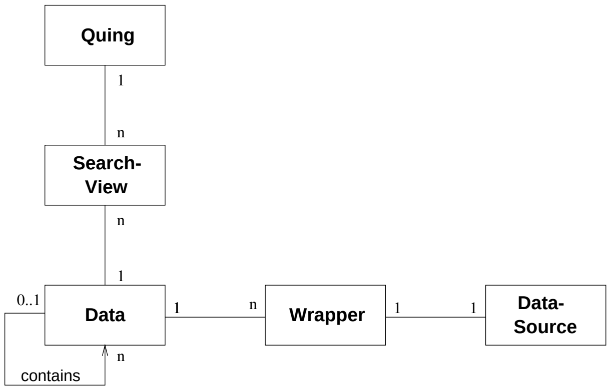

Figure 3: Quing structure (class diagram)

Inside a query, a Quing represents a result set of objects (in the sense of the Quing Data Model). A query consists of one or more Quings which are connected. There are two kinds of connections: Either the result sets of two Quings are put together using a set operator (union, intersection, difference), or the result sets of two Quings are compared by values, which is a kind of a semi-join. For details see [13]. Ideally, there exists a couple of pre-defined Quings for a special application (following the kinds of queries described above, for example).

The definition of the supported search functionality is easy: Appropriate Quings have to be selected. In contrast, the task of writing wrappers and search views is done by experts. If the data is stored in a database management system, the specification of the search functionality of the search view can be done using SQL, OQL, etc. commands, which normally can be carried out by semi professionals with knowledge in the database area.

4.4 Abstract query language

The Quings are the basis for an abstract query language with an abstract syntax. Because of its simplicity there can easily be generated concrete query languages, textual languages as well as graphical languages. In general, the use of abstract syntaxes is very useful because

- all aspects that are not needed within the semantics definition can be safely ignored ([11]) and
- different concrete languages can be specified based on the same query model (and its semantics).

The special use of abstract query languages for the relational data model is described in [10], in [11] an abstract syntax for graphical query languages can be found.
We define a query algebra for defining syntax and semantics of Quings (which is outside the scope of this paper), and based on this an abstract syntax can be defined. This can be portrayed in simplified terms as follows (in EBNF):

<table>
<tr><td>Query</td><td>::=</td><td>Quing |</td><td>(simple query)</td></tr>
<tr><td></td><td></td><td>Query ($\cup \mid \cap \mid \neg\cap$) Query |</td><td>(connection by a set operator)</td></tr>
<tr><td></td><td></td><td>Query $A(\forall \mid \exists)OP\ B$ Query</td><td>(connection by attributes)</td></tr>
<tr><td>OP</td><td>::=</td><td>$= \mid \neq \mid \leq \mid \geq \mid similar \mid \ldots$</td><td>(operators)</td></tr>
<tr><td>Quing</td><td>::=</td><td>QName Param*</td><td>(Quing-Name; parameters)</td></tr>
</table>

(A and B are names of attributes in the Quing Data Model.)
From this abstract syntax we can easily derive concrete syntaxes, for example a SQL like textual and a graphical representation based on HTML forms.
A textual query language is defined following the grammar below (in EBNF):

<table>
<tr><td>Query</td><td>::=</td><td>”SELECT” Q ID (”,” Q ID)*</td></tr>
<tr><td></td><td></td><td>”WHERE” (ID (”AND” | ”OR” | ANDNOT) |</td></tr>
<tr><td></td><td></td><td>(”FORALL” | ”EXISTS”) ID ”.”A OP id ”.” B)</td></tr>
<tr><td>OP</td><td>::=</td><td>”=” | ”<>” | ”similar” | ...</td></tr>
<tr><td>Q</td><td>::=</td><td>QName Param (”,” Param)*</td></tr>
</table>

($QName$ represents a name of one of the available Quings, A and B denote attribute names, ID and $Param$ denote any character strings.)
Let be a Quing $QAttribute$ with two parameters (”AName” and ”AValue”), which serve for searching of pages in virtual courses; the parameter ”AName” denotes the name of the attribute, the parameter ”AValue” denotes the value of an attribute; both search terms can include regular expressions.
The query ”search all pages whith the title ’design pattern’” which include an example can be expressed in this concrete syntax as

```
SELECT QAttribute q1("title","design pattern"),
       QAttribute q2("contenttype","example")
WHERE q1 AND q2
```

The query ”search all pages about ’MVC’ within the chapter with the title ’design pattern’” referring to the structure of virtual courses can be formulated by the following expression (using suitable attributes):

```
SELECT QAttribute q1("content","MVC"),
       QAttribute q2("title","design pattern")
WHERE FORALL q1.parent = q2.oid
```

The search results in all pages which contain ”MVC” (by the attribute ”content” a full-text search is meant) and which posses the pages represented by ”q2” as parents. The attribute ”parent” references to the hierarchical higher page; the attribute ”oid” represents the –unique– object id of a page.
A graphical syntax can be defined informally as follows. Every Quing is represented by `<form>`-fields in an HTML form, the count of the field depends on the count of parameters of the corresponding Quing. The form in Fig. 4 consists of three Quings: one Quing for searching for titles, one for searching for content-types (for example ”example”, ”exercise”, ”definition”, ...) and one for a full-text search. Using these Quings the foremost query above represented in this graphical query syntax is shown in Fig. 4.

5 State of the thesis

With the *Quings* a query model for an abstract query language is developed. The semantics of the Quings are expressed by formulas of the predicate calculus based on the Quing Data Model. The abstract query language serves as a basis for many concrete query languages, textual languages as well as graphical languages. An abstract syntax (see [13]) serves as a basis for concrete syntaxes. In order to be able to represent data from different information sources and to allow a simple specification of queries a simple general data model (*Quing Data Model*, *QDM*) was introduced. These concepts are partly implemented in the area of a virtual course. It turned out that the use of the Quings is very easy and flexible. In our prototype, the data of the virtual course are represented as XML files (metadata) and HTML files (content). In text files additional metadata can be defined. The course is generated by a tool which produces (serialized) Java objects at the same time; these objects contain the metadata using the Quing Data Model representation. This data is materialized (see section 4.3), the Java objects contain the data in the format of the Quing Data Model. In contrast, the data from HTML and text files is transient and will be transformed into the Quing Data Model at runtime. In order to search HTML and text files, wrapper for transforming this data into the Quing Data Model were implemented.
In Fig. 4 a search form for virtual courses is shown. The students can look for pages having specified titles (subtitles, respectively), containing one or more words (full-text search) or representing a special kind of content like ”definition”, ”example”

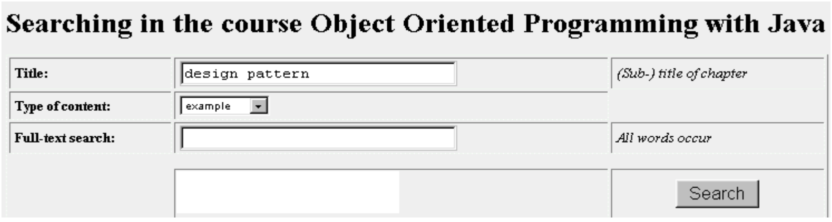

Figure 4: Search in a virtual course

or ”exercise”. Every field of this form represents a special search functionality and is bound to a particular *Quing* internally whereby the Quings are connected by set operators (see section 4).

6 Further work

The definition of a data model (Quing Data Model) and an abstract query language is almost completed. Some of these concepts are already implemented, but much work has to be done. Like in section 3.1 we classify the topics by the three subsections ”data model”, ”query language” and ”implementation”.

6.1 Data model

A language for describing data of the Quing Data Model has to be specified. Presumably, the syntax will be based on XML.
The transformation of data of other data models into the Quing Data Model and vice versa will be investigated. If there arises a loss of information, we will determine if this is acceptable (in the area of virtual courses).

6.2 Query language

With regard to query languages the following tasks have to be considered:

- A language for describing queries with Quings has to be specified (presumably in XML as well). Beside the query itself it should be specified which data source(s) have to be considered.

- The semantics (expressed by formulas of the predicate calculus) is independent of an eventually concrete database management system for storing materialized data (see section 4.3). Possibly, the formulas mentioned above can be translated in the query language of such a database management system.

- Path expressions and recursive queries should be available.

- Quings represent a kind of view. Therefore a discussion of views in other models / systems is necessary, for example views in relational database systems ([9]), *object views* in the Garlic project ([12]) or views for XML ([3]).

- Another interesting aspect is the restructuring of data. For example, searching for exercises in a virtual course results in a set of pages in which only the data relating to exercises are listed, the rest is removed from the original page.

- The expressiveness of the Quing query model has to be investigated. Because there exists no suitable metric, a comparison with other query languages like SQL, *Lore* ([6]) or XML query languages, for example *XQL* ([14]) and *XML-GL* ([8]), is appropriate.

6.3 Implementation

We will implement the Quings for two access alternatives: On the one hand, we will use a relational database management system; in this case the Quing Data Model will be transformed into relations of the database; the queries expressed by Quings have to be converted to SQL commands. On the other hand, no database management system will be used; the Quings use the external data using wrapper classes. The latter is possible because the data only needs to be read and not to be managed. The query processing has to be optimized, this will be done by suitable algorithms integrated in the Quings.
Finally, the implementation will be actually used and evaluated in virtual learning. In order to do this, criteria for evaluation have to be defined.

References

[1] Federal flagship project 'Virtual University of Applied Sciences'. http://www.vfh.de.

[2] Object Data Management Group's Web Page. http://www.odmg.org.

[3] S. Abiteboul. On Views and XML. In *Proc. ACM Symp. on Principles of Database Systems*, pages 1–9, 1999.

[4] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufman, 2000.

[5] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Simeon. Querying Documents in Object Databases. *Journal on Digital Libraries*, 1997.

[6] S. Abiteboul, D. Quass, J. McHugh, Widom J., and J.L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, April 1997.

[7] P. Buneman. Semistructured data. In *PODS'97*, 1997. Invited Tutorial.

[8] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: A Graphical Language for Querying and Reshaping XML Documents. In *QL98 - The Query Languages Workshop*. W3C, 1998.

[9] E.F. Codd. A relational model of data for large shared data banks. In *Comm. of the ACM*, volume 13, pages 377–387, 1970.

[10] C.J. Date. *An Introduction to Database Systems*. Addison-Wesley, 1995.

[11] M. Erwig. Abstract Syntax and Semantic of Visual Languages. *Journal of Visual Languages and Computing*, 9(5):461–483, 1998.

[12] L.M. Haas, R.J. Miller, B. Niswonger, M. Tork Roth, P.M. Schwarz, and E.L. Wimmers. Transforming Heterogenous Data with Database Middleware: Beyond Integration. In *Data Engineering Bulletin*. IEEE, 1999.

[13] C. Lecon, G. Saake, and S. Seehusen. Building Query Interfaces Using Software Engineering Concepts. Technical Report SIA-13, University of Applied Sciences Lübeck, 2000.

[14] J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). In *QL98 - The Query Languages Workshop*. W3C, 1998.

[15] Mary Tork Roth and Peter M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 266–275. Morgan Kaufmann, 1997.

[16] S. Seehusen, C. Lecon, and C. Kaben. Specification of Learning Paths in Virtual Courses. In *Frontiers in Education (FIE'2000)*, Kansas City, October 2000.