Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation

C. Baru, B. Ludäscher, Y. Papakonstantinou, P. Velikhov, V. Vianu

1. Introduction

XML indicates a move towards viewing the Web as a big semistructured database, consisting of multiple autonomous sites which will be modeled around (guess what?) XML and sibling standards for

- structure and ontology definitions (XML-Data, RDF, DCD, namespaces),
- APIs (most probably some extension of DOM),
- source query capability specifications (see below),
- other standards that may describe the transactional abilities of the sites, and so on.

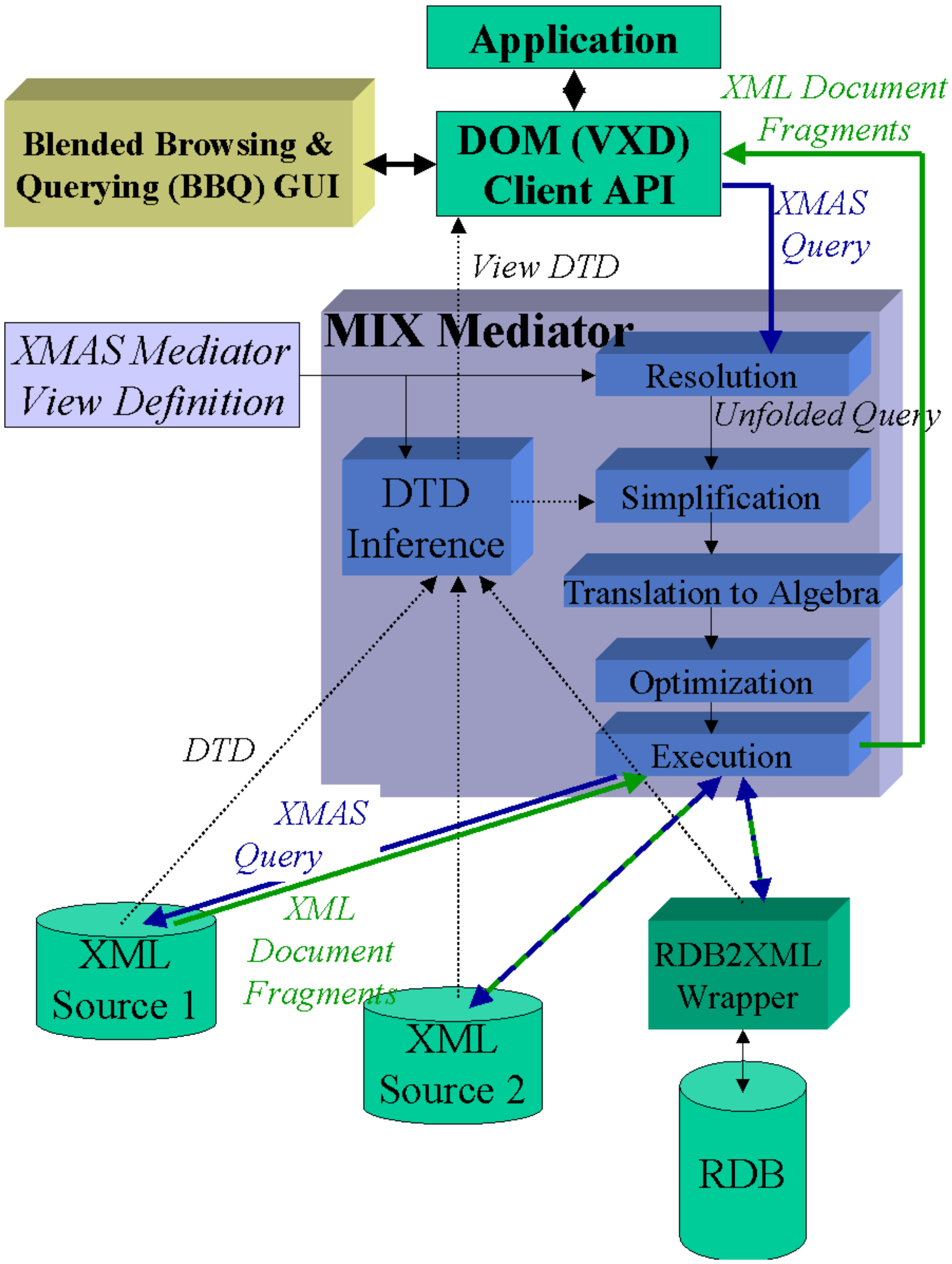
In such an environment a query language will serve for more than "a souped-up version of X-Pointer". Our position focuses on the use of an XML query language as a *view definition* language that drives XML mediator systems. A mediator selects, restructures and merges information from multiple autonomous sources/sites. It exports an integrated XML view document. Possible applications of such mediator systems are numerous (e.g., virtual shopping malls, virtual agencies, ...).

Section 2 gives a brief presentation of the under development *MIX (**M**ediation of **I**nformation using **X**ML)* mediator system in order to illustrate the architecture and the functionality of a typical mediator system. Section 3 presents a list of issues and challenges that we have found in the course of working on the MIX project and the *XMAS (**X**ML **M**atching **A**nd **S**tructuring)* view definition language.

2. The Architecture and Functionality of the MIX Mediator System

The figure below illustrates the architecture of the MIX mediator system [1]. The system accesses a set of XML sources, which are currently information systems wrapped to provide (1) an XML view of their data and (2) a set of XMAS queries that they can answer. (It is likely that in the future there will be XML databases that require no wrapping.) Conceptually, MIX exports an integrated view of the source data. The view definition, provided by the view developer, specifies how the source data will be integrated. Notice that the MIX does not materialize the integrated view in advance.

During runtime, an application (which may be the *BBQ* GUI [1]) issues a XMAS query against the integrated view. The mediator decomposes the client query into queries that are routed to the XML sources. The sources generate and send to the mediator the query results, which are XML documents. The mediator integrates them into the client query result document.



Note that the client accesses the query result using our home-grown version of DOM, called *DOM-VXD* (***DOM*** *for* ***V****irtual* ***X****ML* ***D****ocuments*) . DOM-VXD does not require a complete copy of the query result to be materialized at the client site. In this way we can pipeline the computation and delivery of the query result document and even navigate into it.

3. What we have for XMAS and What we'd like a Santa "QL Committee" Claus to bring us

There are numerous desiderata for a general purpose XML QL; see for an excellent overview. Here, we focus on first lessons learned from designing and implementing the XMAS view definition language for information mediation.
A recurring theme of the following discussion is that expressive power comes at a price which may be:

- inefficient query processing,
- complex query processors,
- inability to perform type inference, etc.

XMAS currently leans towards a conservative database approach thereby delivering the benefits described in the sections below. The cost is some limitations in the expressive power.
Undoubtedly, more powerful features (e.g., see Section 3.6); will also have to be encompassed in XMAS (and, in general, any XML view definition languages). The language should allow for; "graceful" scaling of the expressive power -- a task that will not always be easy.

3.1 XML Faithfulness

An XML view definition should be *faithful* to XML, i.e., it should always produce XML output. The immediate benefit for mediation is that the view is no different from any other XML document.
An ideal solution would be to have a language that can never produce non-XML output. However, this approach may compromise the expressive power of the language. A more viable solution is to develop static analysis tools that can notify the view designer of potential errors, i.e., cases where non-XML output may be produced. Note that by controlling; the complexity of the language we will be able to develop effective static analysis tools.
XMAS takes a middle-of-the-road approach: Its ability to move data from a source's content to element names or attribute names creates the possibility of non-XML output. This instance of potential unfaithfulness can be caught using a trivial static safety check.
On the other hand, XMAS avoids many sources of potential unfaithfulness. For example XMAS has avoided (insofar) the use of Skolem functions, which can be a major source of unfaithfulness. For example, they make it difficult to check whether attributes of elements are unique. Nevertheless, Skolem functions can deliver expressive power that cannot always be substituted by the SQL-like group-by mechanism of XMAS.

3.2 Type Inference

Given descriptions of the source documents (such as DTDs, XML-Data specifications, or DCDs), the mediator should be able to compute a correct and precise description of the view. We have developed algorithms [1,2] that compute precise view DTDs from the source DTDs and the view definition.
Again, the expressive power versus complexity tradeoff is encountered: DTD inference is not possible for views that move data from a source's content to element names or attribute names; such views are allowed in XMAS. On the other hand, the explicit group-by operator allows us to derive precise types.
A closely related property is *type conformance* of a view definition to a predefined DTD. Similar challenges and tradeoffs with type inference appear in this case as well.

3.3 Closure Under Composition

A typical task for query processors (and not only XML ones) is the following: Given (1) a client query q that refers to a view V and (2) the query which defines V in terms of some source database(s) S, derive a query q' that is equivalent to q and refers directly to S. It is important for the simplicity and efficiency of query processing that q' is expressed in the same language as q and V. That is, the language should be *closed under query composition.*

It is important to note that closure under composition requires careful design of the query language. For example, one can show that a query language that allows Skolem object-ids and regular path expressions but does not allow any other form of recursion, is not closed under composition.

In XMAS, closure under composition is achieved by employing a group-by construct instead of Skolem functions. Once again, a compromise in expressiveness leads to a desired property.

3.4 Order: Manipulating, Preserving, and Never-Minding it

XML has order. Elements are organized in lists, as opposed to sets. However applications, in general, have different requirements on the order issue. An XML query language should address the following issues under a single semantics:

- **Order Preservation:** By default the element order of the input is preserved.
- **Order Manipulation** The view designer is able to define a new order of elements. A typical task is the ordering of elements using a ranking function (e.g., in the style of SQL's order-by). Once again notice that extremely powerful order manipulation mechanisms will compromise type inference and closure under composition.
- **Order Nondeterminism:** Finally the view designer may want to specify that order in the view is unimportant. In this case he can specify that the order is nondeterministic thereby leaving room for optimization.

The above policies deal with specifying an order for the view. Similar considerations apply to checking the order of the input elements. In this case XMAS' default is "don't care" about the order of elements in the input. Conversely, if the view/query has to put a condition on the order of input elements an *horizontal navigation regular expression* can be used [2].

3.5 Query Processing: Put an Algebra in Your System

In order to apply similar optimization techniques as those developed for relational databases, an XML query language should have an equivalent algebra. An algebra will also facilitate the implementation of pipelining/browsing as described in Section 2.

3.6 All Processing in One Language: The Challenge of Structural Recursion

It is desirable to have a language that tackles all tasks required in the mediation process. In this way we can avoid the impedance mismatch caused by using multiple languages and we create extra opportunities for optimization.
It seems that two language paradigms have emerged in the XML querying context and they will have to be reconciled:

- The database paradigm has an underlying logic semantics and is convenient for selecting and integrating objects. It has recently been extended to allow navigational recursion in a clean way. However it does not support (without the use of hard-to-process general recursion) restructuring of "deep" structures.

- The functional paradigm, exemplified by XSL and now XQL, is particularly suitable for "deep" transformations of XML documents. However it is cumbersome or less powerful when it comes to conventional search and join.

[UnQL] shows a promising direction in bridging the two paradigms.

3.7 Describing the Query Capabilities of Sources

XML sources will only be able to support a subset of the queries over the "schema" they export. For example, consider a site that exports a flight information schema (say, DTD) but it only accepts queries that specify a destination, an arrival, and a optional price range. It is clear that a mediator system or, in general, any agent that queries this source must conform to the set of supported queries exported by this source.
So it is important that mechanisms are developed for describing the set of queries that are supported by a site [3].

3.8 Metadata

In communities such as, decision support, OLAP, and scientific databases, the term "metadata" refers to auxiliary information associated with defined data sets. For example, this may include attributes that provide further information about the data, annotations/explications associated with the data, and/or computed/derived information related to the data. In general, whether a piece of information is data or metadata depends on one's point of view. In defining a virtual site, the view specification mechanism must provide the capability to specify the data as well as the metadata for that site. Once this distinction between data and metadata has been made at the site level, this information can be used in a variety of ways in the system. For example, it can be used at the interface level to support different query and presentation modes for data vs. metadata, and it can be used at the data transport level to encode data differently than metadata.

4. Conclusions

Information mediation poses a challenging set of requirements to an XML query and view definition language. We discuss above several important issues: XML faithfulness, type inference, order, closure under composition, etc. A recurring theme is the search for the right balance between expressive power, on the one side, and efficiency and viability on the other.