Intelligent Querying of Web Documents Using a Deductive XML Repository

[§]Nick Bassiliades, Ioannis Vlahavas

Dept. of Informatics
Aristotle University of Thessaloniki
54006 Thessaloniki, Greece
{nbassili,vlahavas}@csd.auth.gr

Abstract
In this paper, we present a deductive object-oriented database system, called X-DEVICE, WHICH is used as a repository for XML documents. X-DEVICE employs a powerful rule-based query language for intelligently querying stored Web documents and data and publishing the results. XML documents are stored into the OODB by automatically mapping the DTD to an object schema. XML elements are treated either as objects or attributes based on their complexity, without loosing the relative order of elements in the original document. The rule-based language features second-order logic syntax, generalized path and ordering expressions, which greatly facilitate the querying of recursive, tree-structured XML data and the construction of XML trees as query results. All the extended features of the rule language are translated through the use of object metadata into a set of first-order deductive rules that are efficiently executed against the object database using the system's basic inference engine.

Keywords: XML, Object Database, Deductive Database
Conference Topics: Knowledge Representation, Logic Programming, Intelligent Systems

Περίληψη
Η εργασία αυτή παρουσιάζει ένα συνεπαγωγικό, αντικειμενοστραφές σύστημα βάσεων δεδομένων, το X-DEVICE, το οποίο χρησιμοποιείται για αποθήκευση XML εγγράφων. Στο σύστημα X-DEVICE υπάρχει μία ισχυρή γλώσσα ερωταπαντήσεων η οποία βασίζεται σε κανόνες και χρησιμοποιείται για τη διατύπωση έξυπνων ερωτημάτων σχετικά με τα αποθηκευμένα έγγραφα και δεδομένα ιστοσελίδων, καθώς και για τη δημοσίευση των αποτελεσμάτων. Τα XML έγγραφα αποθηκεύονται αυτόματα στην αντικειμενοστραφή βάση δεδομένων με αντιστοίχιση του αντίστοιχου εγγράφου DTD που περιγράφει τη δομή τους σε δομή αντικειμένων. Τα στοιχεία XML του αρχικού εγγράφου αντιμετωπίζονται είτε ως αντικείμενα ή χαρακτηριστικά αντικειμένων, ανάλογα με την πολυπλοκότητά τους, χωρίς να χάνεται η σχετική διάταξη μεταξύ τους. Η γλώσσα των κανόνων χαρακτηρίζεται από σύνταξη λογικής δεύτερης τάξης, γενικευμένες εκφράσεις μονοπατιών αναζήτησης, καθώς και εκφράσεις διάταξης. Τα χαρακτηριστικά αυτά διευκολύνουν τη διατύπωση ερωτημάτων πάνω στα δεδομένα XML, τα οποία έχουν αναδρομική, δενδροειδή δομή, καθώς και τη δημιουργία αποτελεσμάτων υπό τη μορφή δενδροειδών δομών. Όλα τα παραπάνω χαρακτηριστικά μεταφράζονται με τη βοήθεια των μετα-δεδομένων της αντικειμενοστραφούς βάσης σε ένα σύνολο κανόνων λογικής πρώτης-τάξης, οι οποίοι αποτιμούνται πάνω στα δεδομένα της βάσης μέσω της βασικής μηχανής εξαγωγής συμπερασμάτων του συστήματος.

[§] Supported by a post-doctoral scholarship from the Greek Foundation of State Scholarships (F.S.S. - I.K.Y.).

1. Introduction

Currently information is captured and exchanged over Internet through HTML pages, without any conceptual structure. XML is the currently proposed standard for structured or even semi-structured information exchange over the Internet. However, the maintenance of the information captured from XML documents is essential for building long-lasting applications of industrial strength. The enormous research and development of DBMSs should be re-used for managing XML data with the minimum of effort. There already exist several proposals on methodologies for storing, retrieving and managing XML data stored in relational and object databases.

Another important aspect of managing XML is effective and efficient querying and publishing these data on the Web. There have been several query language proposals ([11], [2], [9]) for XML data. Furthermore, recently the XML Query Working Group [24] of the WWW consortium issued a working draft proposing XQuery, an amalgamation of the ideas present in most of the proposed XML query languages of the literature. Most of them have functional nature and use path-based syntax. Some of them, including XQuery, have also borrowed an SQL-like declarative syntax, which is popular among users. Some of the problems relating to most of the above approaches is the lack of a comprehensible data model, a simple query algebra and query optimization techniques. There are proposals for a data model and a query algebra for XQuery, however it is not yet clear how these will lead to efficient data storage and query optimization.

In this paper, we present a deductive object-oriented database system, called X-DEVICE, WHICH is used as a repository for XML documents. X-DEVICE employs a powerful rule-based query language for intelligently querying stored Web documents and data and publishing the results. The advantages of using a logic-based query language come from their well-understood mathematical properties. The declarative character of these languages also allows the use of advanced optimization techniques.

X-DEVICE is an extension of the active object-oriented knowledge base system DEVICE ([4]). DEVICE integrates high-level, declarative rules (namely deductive and production rules) into an active OODB that supports only event-driven rules [13], built on top of Prolog. This is achieved by translating each high-level rule into one event-driven rule. The condition of the declarative rule compiles down to a set of complex events that is used as a discrimination network that incrementally matches the rule conditions against the database.

In X-DEVICE, XML documents are stored into the OODB by automatically mapping the DTD to an object schema. XML elements are treated either as objects or attributes based on their complexity, without loosing the relative order of elements in the original document. The rule-based language features second-order logic syntax, generalized path and ordering expressions, which greatly facilitate the querying of recursive, tree-structured XML data and the construction of XML trees as query results. All the extended features of the rule language are translated through the use of object metadata into a set of first-order deductive rules that are efficiently executed against the object database using the system's basic inference engine. The formal translation procedures have been presented elsewhere [6]. In this paper we mainly focus on the use of the X-Device query language on intelligently querying XML documents.

The outline of this paper is as follows: in section 2 we overview some of the related work done in the area of storing and querying XML data in databases; Section 3 describes the mapping of XML data onto the object data model of X-DEVICE; Section 4 presents the X-DEVICE deductive rule language for querying XML data through several examples that have been adopted from the XML Query working group ([24]). Finally, Section 0 concludes this paper and discusses future work.

2. Related Work

There exist two major approaches to manage and query XML documents. The first approach uses special purpose query engines and repositories for semi-structured data [15], [20], [21]. These database systems are built from scratch for the specific purpose of storing and querying XML documents. This approach, however, has two potential disadvantages. Firstly, native XML database systems do not harness the sophisticated storage and query capability already provided by existing database systems. Secondly, native XML database systems do not allow users to query seamlessly across XML documents and other (structured) data stored in database systems. The second approach captures and manages XML data within the data models of either relational ([23], [12], [14]) or object databases ([25], [22], [10]). Our system, X-DEVICE, stores XML data into the object database ADAM [16], because XML documents have by nature a hierarchical structure that better fits the object model. Also references between or within documents play an important role and are a perfect match for the notion of object in the object model.

When XML data are mapped onto relations there are some limitations: First, the relational model does not support set-valued attributes, therefore when an element has a multiply-occurring sub-element, the sub-element is made into a separate relation and the relationship between the element and the sub-element is represented by a foreign key. The querying and reconstruction of the XML document requires the use of "expensive" SQL joins between the element and sub-element relations. On the other hand, object databases support list attributes; therefore, references to sub-elements can be stored with the parent element and retrieved in a non-expensive way. A second limitation of relational databases is that in order to represent relationships between elements-relations, join attributes should be created manually. On the other hand, in object databases, relationships between element-classes are represented in the schema by object-referencing attributes (pointers), which are followed for answering queries with path expressions. Finally, relations are sets with no ordering among their rows or columns. However, in XML documents ordering of elements is important, especially when they contain textual information (e.g. books, articles, Web page contents). There exist some relational approaches that hold extra ordering information in order to be able to reconstruct the original XML documents, which add extra complication to the relation schema, query processing and XML reconstruction algorithms.

Object database approaches usually treat element types as classes and elements as objects. Attributes of elements are treated as text attributes, while the relationships between elements and their children are treated as object referencing attributes. There are some variations of the above schema between the various approaches. For example, in [1] and [25] all the elements are treated as objects, even if their content is just PCDATA, i.e. mere strings. However, such a mapping requires a lot of classes and objects, which wastes space and degrades performance, because queries have to traverse more objects than actually needed. In X-DEVICE this problem is avoided by mapping PCDATA elements to text attributes.

The in-lining approach that has been proposed in [23] for relational databases is followed in [10] to avoid producing too many classes in the schema. However, the rationale for the in-lining method in [23] was that it reduces the amount of tables and joins between them, while in object databases there are no joins and therefore there is no rationale for using it. Furthermore, the resolution of path expressions gets complicated since some parts of the path consist of simple attribute names, while some others consist of path fragments that are "in-lined" as a single attribute in a great master class. The decision on which parts are simple and which complex is based on 5 rules.

Another major issue that must be addressed by any mapping scheme is the handling of the flexible and irregular schema of XML documents that includes alternation elements. Some mapping schemes, such as [23], avoid handling alternation by using some simplification rules, which transform alternation to sequence of optional elements: `(X|Y)->(X?,Y?)`. However, this transformation, along with some other ones, does not preserve equivalence between the original and the simplified document. In the previous simplification rule, for example, the element declaration on the left-hand side accepts either an `X` or a `Y` element, while the right-hand side element declaration allows also a sequence of both elements or the absence of both.

Alternation is handled by union types in [1], which required extensions to the core object database O_2. This approach is efficient, however it is not compatible with the ODMG standard and cannot easily be applied in other object database system. In X-DEVICE instead of implementing a union type, we have emulated it using a special type of system-generated class that its behavior does not allow more than one of its attributes to have a value. Furthermore, the parent-element class hosts aliases for this system generated class, so that path resolution is facilitated.

Logic has been used for querying semi-structured documents, in WebLog [17] for HTML documents and in F-Logic [18] and XPathLog [19] for XML data. WebLog is a deductive Web querying language, similar in spirit to Datalog, operating in an integrated graph-based framework. Although its syntax resembles F-Logic, it is not fully object-oriented. Navigation along hyperlinks is provided by built-in predicates, but navigation in the form of path expressions is not supported. Furthermore, there is no notion of generalized path expressions. Instead, recursive Datalog-like rules replace regular expressions over attributes, attribute variables, and path variables. X-DEVICE offers both regular path expressions and generalized path expressions, i.e. path expressions with unknown number and names of intermediate steps.

F-Logic and its successor XPathLog are logic-based languages for querying semi-structured data. Their semantics are defined by bottom-up evaluation, similarly to X-DEVICE, however negation has not been implemented. Both languages can express multiple views on XML data; XPathLog, in addition, can export the view in the form of an XML document, much like X-DEVICE. However, none of the languages offers incremental maintenance of materialized views when XML base data get updated, as X-DEVICE does.

Both F-Logic and XPathLog are based on a graph data model, which can be considered as a schema-less object-oriented (or rather frame-based) data model. However, alternation of elements is not supported. Furthermore, F-Logic does not preserve the order of XML elements. Both languages support path expressions and variables; XPathLog is based on the XPath language syntax [24]. The main advantage of both languages is that, similar to X-DEVICE, they can have variables in the place of element and/or attribute names, allowing the user to query without an exact knowledge of the underlying schema. However, none of the languages supports generalized path expressions that X-DEVICE does, which compromises their usefulness as semi-structured query languages.

3. The Object Model of XML Data

The X-DEVICE system incorporates XML documents with a schema described through a DTD into an object-oriented database. Specifically, XML documents (including DTD definitions) are fed into the system. DTD definitions are translated into an object database schema that includes classes and attributes, while XML data are translated into objects of the database. Generated classes and objects are stored within the underlying object-oriented database ADAM ([16]).

Concerning the XML Query Data Model ([24]), X-DEVICE currently maps a subset of the model's node types, namely: document, element, value, attribute, and node reference. The mapping between a DTD and the object-oriented data model is done as follows:

- The *document node* type is represented by the `xml_doc` class and each document node is an instance of this class. The attributes of this class include the URI reference of the document and the OIDs of its root element nodes.
- *Element nodes* are represented as either object attributes or classes. More specifically:
    - ➢ If an element node has PCDATA content (without any attributes), it is represented as an attribute of the class of its parent element node. The name of the attribute is the same as the name of the element and its type is string.
    - ➢ If an element node has either a) children element nodes, or b) attribute nodes, then it is represented as a class that is an instance of the `xml_seq` meta-class. The attributes of the class include both the attributes of the element and the children element nodes. The types of the attributes of the class are determined as follows:
        - Simple character children element nodes correspond to attributes of string type.
        - Attribute nodes correspond to attributes of string type.
        - Children elements that are represented as objects correspond to object reference attributes.
- *Value nodes* are represented by the values of the object attributes. Currently, only strings and object references are supported since DTDs do not support data types.
- *Attribute nodes* are represented as object attributes. For the types of the attributes, the status is the same as for the value nodes. Attributes are distinguished from children elements through the `att_lst` meta-attribute.
- *Node references* are represented as object references.

In Appendix A there is an examples of the OODB schema that is generated using our mapping scheme, for the DTD of the "TEXT" XML Query Use Case that can be found in [24].

There are more issues that a complete mapping scheme needs to address, except for the above mapping rules. First, elements in a DTD can be combined through either sequencing or alternation. *Sequencing* means that a certain element must include *all* the specified children elements with a specified order. This is handled by the above mapping scheme through the existence of multiple attributes in the class that represents the parent element, each for each child element of the sequence. The order is handled outside the standard OODB model by providing a meta-attribute (`elem_ord`) for the class of the element that specifies the correct ordering of the children elements. This meta-attribute is used when (either whole or a part of) the original XML document is reconstructed and returned to the user. The query language also uses it, as it will be shown later.

On the other hand, *alternation* means that *any* of the specified children elements can be included in the parent element. Alternation is also handled outside the standard OODB model by creating a new class for each alternation of elements, which is an instance of the `xml_alt` meta-class and it is given a system-generated unique name. The attributes of this class are determined by the elements that participate in the alternation. The types of the attributes are determined as in the sequencing case. The structure of an alternation class may seem similar to a sequencing class, however the behavior of alternation objects is different, because they must have a value for exactly one of the attributes specified in the class.

The alternation class is always encapsulated in a parent element. The parent element class has an attribute with the system-generated name of the alternation class, which should be hidden from the user for querying the class. Therefore, a meta-attribute (`alias`) is provided with the aliases of this system-generated attribute, i.e. the names of the attributes of the alternating class. Mixed content elements are handled similarly to alternation of elements. The only difference is that one of the alternative children elements can also be plain text (PCDATA), which is handled by creating a string attribute of the alternation class, called `'$content'`.

Another issue that must be addressed is the mapping of the occurrence operators for elements, sequences and alternations. More specifically, these operators are handled as follows:

- The "star"-symbol (*) after a child element causes the corresponding attribute of the parent element class to be declared as an optional, multi-valued attribute.
- The "cross"-symbol (+) after a child element causes the corresponding attribute of the parent element class to be declared as a mandatory, multi-valued attribute.
- The question mark (?) after a child element causes the corresponding attribute of the parent element class to be declared as an optional, single-valued attribute.
- Finally, the absence of any symbol means that the corresponding attribute should be declared as a mandatory, single-valued attribute.

The order of children element occurrences is important for XML documents, therefore the multi-valued attributes are implemented as lists and not as sets.

Empty elements are treated in the above framework, depending on their internal structure. If an empty element does not have attributes, then it is treated as a PCDATA element, i.e. it is mapped onto a string attribute of the parent element class. The only value that this attribute can take is `yes`, if the empty element is present. If the empty element is absent then the corresponding attribute does not have a value. On the other hand, if an empty element has attributes, then is represented by a class. Finally, unstructured elements that have content `ANY` are not currently treated by X-DEVICE.

4. The Deductive XML Query Language

Currently users can query the stored XML documents using X-DEVICE by entering the query directly in the text-based Prolog environment. The X-DEVICE query is transformed into the basic DEVICE rule language and is executed using the system's basic inference engine. The query results are returned to the user in the form of an XML document.

The deductive rule language of X-DEVICE supports constructs and operators for traversing and querying tree-structured XML data, which are implemented using second-order logic syntax (i.e.

variables can range over class and attribute names) that can also been used to integrate heterogeneous schemata [5]. These XML-aware constructs are translated into a combination of a) a set of first-order logic deductive rules, and/or b) a set of production rules that their conditions query the meta-classes of the OODB, they instantiate the second-order variables, and they dynamically generate first-order deductive rules.

Throughout this section, we will demonstrate the use of X-DEVICE for querying Web documents in XML format using examples taken from the "TEXT" XML Query Use Case proposed by the XML Query Working Group ([24]). This use case is based on company profiles and a set of news documents which contain data for PR, mergers and acquisitions, etc. Given a company, the use case illustrates several different queries for searching text in news documents and different ways of providing query results by matching the information from the company profile and the content of the news items. The X-DEVICE object schema for this case is shown in Appendix A.

In this section, we give a brief overview of the X-DEVICE deductive rule language. More details about DEVICE and X-DEVICE can be found in [4], [5], [6]. Especially, the general algorithms for the translation of the various XML-aware constructs to first-order logic have been presented in [6]. Here, due to space limitations, we will present only few translation cases, so that the reader can have an idea of the process.

First-Order Deductive Query Language

In X-DEVICE, deductive rules are composed of condition and conclusion, whereas the condition defines a pattern of objects to be matched over the database and the conclusion is a derived class template that defines the objects that should be in the database when the condition is true. The following rule defines that an object with attribute `partner=P` exists in class `partner_of_xyz` if there is an object with OID `C` in class `company` with an attribute `name='XYZ Ltd'` and an attribute `partners` which points to an object of class `partners` which in turn has an attribute `partner=P`.

```
if C@company(name='XYZ Ltd',partner.partners ∍ P1)
then partner_of_xyz(partner:P1)
```

Class `partner_of_xyz` is a derived class, i.e. a class whose instances are derived from deductive rules. Only one derived class template is allowed at the THEN-part (head) of a deductive rule. However, there can exist many rules with the same derived class at the head. The final set of derived objects is a union of the objects derived by the two rules. For example, the transitive closure of the set of direct and indirect partners of company 'XYZ Ltd' is completed with the following (recursive) rule:

```
if P@partner_of_xyz(partner:P1) and
   C@company(name=P1,partner.partners ∍ P2)
then partner_of_xyz(partner:P2)
```

The syntax of such a rule language is first-order. Variables can appear in front of class names (e.g. `P`, `C`), denoting OIDs of instances of the class, and inside the brackets (e.g. `P1`, `P2`), denoting attribute values (i.e. object references and simple values, such as integers, strings, etc). Variables are instantiated through the ':' operator when the corresponding attribute is single-valued, and the '∍' operator when the corresponding attribute is multi-valued. Since multi-valued attributes are implemented through lists (ordered sequences) the '∍' operator guarantees that the instantiation of variables is done in the predetermined order stored inside the list. Conditions also can contain comparisons between attribute values, constants and variables. Negation is also allowed if rules are safe, i.e. variables that appear in the conclusion must also appear at least once inside a non-negated condition.

The path expressions are composed using dots between the "steps", which are attributes of the interconnected objects, which represent XML document elements. The innermost attribute should be an attribute of "departing" class, i.e. `partners` is an attribute of class `company`. Moving to the left, attributes belong to classes that represent their predecessor attributes. Notice that we have adopted a right-to-left order of attributes, contrary to the C-like dot notation that is commonly assumed, because we would like to stress out the functional data model origins of the underlying ADAM

OODB [16]. Under this interpretation the chained "dotted" attributes can be seen as function compositions.

A query is executed by submitting the set of stratified rules (or logic program) to the system, which translates them into active rules and activates the basic events to detect changes at base data. Data then are forwarded to the rule processor through a discrimination network (much alike in a production system fashion). Rules are executed with fixpoint semantics (semi-naive evaluation), i.e. rule processing terminates when no more new derivations can be made. Derived objects are materialized and are either maintained after the query is over or discarded on user's demand. X-DEVICE also supports production rules, which have at the THEN-part one or more actions expressed in the procedural language of the underlying OODB [16].

The main advantage of the X-DEVICE system is its extensibility that allows the easy integration of new rule types as well as transparent extensions and improvements of the rule matching and execution phases. The current system implementation includes deductive rules for maintaining derived and aggregate attributes. Among the optimizations of the rule condition matching is the use of a RETE-like discrimination network, extended with re-ordering of condition elements, for reducing time complexity and virtual-hybrid memories, for reducing space complexity [3]. Furthermore, set-oriented rule execution can be used for minimizing the number of inference cycles (and time) for large data sets [4].

Generalized Path Expressions

X-DEVICE supports several types of path expressions into rule conditions. The previous example demonstrated the simplest case, where all the steps of the path are known. Another case in path expressions is when the number of steps in the path is determined, but the exact step name is not. In this case, a variable is used instead of an attribute name. This is demonstrated by the following example, which searches for companies that contain in any of their immediate (PCDATA) children elements a specified string.

```
if C@company(A $ 'XYZ')
then a_xyz_company(company:list(C))
```

The ($) operator searches its right-hand-side argument (string `'XYZ'`) inside its left-hand-side argument (a string attribute `A`). The `list(C)` construct in the rule conclusion denotes that the attribute `company` of the derived class `a_xyz_company` is an attribute whose value is calculated by the aggregate function `list`. This function collects all the instantiations of the variable `C` (since many companies can contain the string '`XYZ`' in any of their string attributes) and stores them under a strict order into the multi-valued attribute `company`. More details about the implementation of aggregate functions in X-DEVICE can be found in [4].

Variable `A` is in the place of an attribute name, therefore it is a second-order variable, since it ranges over a set of attributes, and attributes are sets of things (attribute values). Deductive rules that contain second-order variables are always translated into a set of rules whose second-order variable has been instantiated with a constant. This is achieved by generating production rules, which query the meta-classes of the OODB, instantiate the second-order variables, and generate deductive rules with constants instead of second-order variables [5]. The above rule is translated as follows:

```
if company@xml_seq(elem_order ∍ A)
then new_rule('if C@company(A $ 'XYZ')
               then a_xyz_company(company:list(C))')
     => deductive_rule
```

Notice that variable `A` is now a first-order variable in the condition of the production rule, while the deductive rule generated by the action of the production rule has `A` instantiated. The above rule will actually produce as many deductive rules, as many attribute names there are in class `company`. The result consists of the union of the results of all the deductive rules.

The most interesting case of path expressions is when some part of the path is unknown, regarding both the number and the names of intermediate steps. This is handled in X-DEVICE by using the "star" (*) operator in place of an attribute name. Such path expressions are called "generalized". The previous example can be re-written using the "star" (*) operator as:

```
if C@company(* $ 'XYZ')
```

```
then a_xyz_company(company:list(C))
```

However, the semantics of the above query are significantly different, since now the search for the `'XYZ'` string is not done only at the immediate children elements of `company`, but on any element of the XML sub-tree that starts from the `company` element.

Ordering Expressions

X-DEVICE supports expressions that query an XML tree based on the ordering of elements. The following query, taken from the "TEXT" XML Query Use Cases in [24], demonstrates X-DEVICE's absolute numeric ordering expressions.

TEXT Case - Q5. For each news item that is relevant to the Gorilla Corporation, create an "item summary" element. The content of the item summary is the content of the title, date, and first paragraph of the news item, separated by periods. A news item is relevant if the name of the company is mentioned anywhere within the content of the news item.

The above query is expressed in X-DEVICE as:

```
if N@news_item(title:T,*.content$'Gorilla Corporation',par.content ∍1 PAR,date:D)
then item_summary(title:T,date:D,par:PAR)
```

The $\ni_1$ operator (a shortcut notation for the $\ni_{=1}$ operator) is an absolute numeric ordering expression that returns the first element of the corresponding list-attribute. More such ordering expressions in X-DEVICE exist for every possible position inside a multi-valued attribute [6]. The ordering expression part of the above rule is translated as follows:

```
if N@news_item(title:T,*.content$'Gorilla Corporation',par.content ∍ XX1,date:D)
then tmp_elem1(tmp_var1:T,tmp_var2:D,tmp_obj:list(XX1))
if XX3@tmp_elem1(tmp_var1:T,tmp_var2:D,tmp_obj:XX1) and
   prolog{select_sub_list('='(1),XX1,XX2)}
then tmp_elem2(tmp_var1:T,tmp_var2:D,tmp_obj:XX2)
if XX1@tmp_elem2(tmp_var1:T,tmp_var2:D,tmp_obj ∍ PAR)
then item_summary(title:T,date:D,par:PAR)
```

The first rule collects all the paragraphs that satisfy the condition, the second rule isolates a sub-list of all the paragraphs that satisfy the ordering expression (through the use of a Prolog goal), and the third rule actually iterates over all qualifying results. Using Prolog goals in the rule condition, the system can be extended with several new features. However these features are outside of the deductive rule language and, therefore, cannot be optimized.

Exporting Results

So far, only the querying of existing XML documents through deductive rules has been discussed. However, it is important that the results of a query can be exported as an XML document. This can be performed in X-DEVICE by using some directives around the conclusion of a rule that defines the top-level element of the result document. When the rule processing procedure terminates, X-DEVICE employs an algorithm that begins with the top-level element designated with one of these directives and navigates recursively all the referenced classes constructing a result in the form of an XML tree-like document [6].

The following example demonstrates how XML documents (and DTDs) are constructed in X-DEVICE for exporting them as results.

TEXT Case - Q6. Find news items where two company names and some form of the word "acquire" appear in the title or in the same sentence in one of the paragraphs. A company name is defined as the content of a <name>, <partner>, or <competitor> element within a <company> element.

The above query is expressed in X-DEVICE as the following logic program:

```
R1: if C@company(name:Name)
    then company_names(name:Name)
R2: if C@company(partner.partners ∍ Name)
    then company_names(name:Name)
R3: if C@company(competitor.competitors ∍ Name)
    then company_names(name:Name)
R4: if C1@company_names(name:Name1) and
       C2@company_names(name:Name2\=Name1) and
```

```
        N@news_item(title:T) and
        prolog{contains_stems_in_same_sentence(T,Name1,Name2,'acquire')}
     then result(news_item:list(N))
R5: if C1@company_names(name:Name1) and
        C2@company_names(name:Name2\=Name1) and
        N@news_item(par.content ∍ PAR) and
        prolog{contains_stems_in_same_sentence(PAR,Name1,Name2,'acquire')}
     then xml_result(result(news_item:list(N)))
```

Rules `R1` to `R3`, in the above program, iterate over all company elements, their partners and competitors and store their names in an auxiliary derived class, called `company_names`. Notice that the same company name is not stored twice in `company_names` because the semantics of derived classes require that not two objects with the exactly the same attribute values should exist [4]. Rules `R4` an `R5` take the Cartesian product of all identified company names and try to establish an acquisition relationship between them either in the title or in any of the paragraphs of a news item. This is achieved through the use of `contains_stems_in_same_sentence/4`, a user-defined Prolog predicate.

The keyword `xml_result` is a directive that indicates to the query processor that the encapsulated derived class (`result`) is the answer to the query. This is especially important when the query consists of multiple rules, as the above example. Notice that although both `R4` and `R5` rules refer to the derived class `result`, only one of them contains the `xml_result` directive. However, this is not a strict language rule; it does not matter if several rules contain the `xml_result` or any other result directive ([6]), as long as the following constraints are satisfied:

- Only one type of result directive is allowed in the same query.
- Only one derived class is allowed at the result.

In order to build an XML tree as a query result, the objects that correspond to the elements must be constructed incrementally in a bottom-up fashion, i.e. first the simple elements that are towards the leaves of the tree are generated and then they are combined into more complex elements towards the root of the tree. Another way to generate an XML tree as a result is to include in the result parts of the (or even the whole) original XML document, as it is the case in the current example. The above query produces a tree-structured XML document, with the following DTD:

```
<!DOCTYPE result [
   <!ELEMENT result (news_item*)>
   <!ELEMENT new_item ...
   ...                          ]>
```

Notice how the `list` aggregate function is used to construct XML elements with multiple children. The definition for the `news_item` element is exactly the one found at the original XML document of the "TEXT" XML Query Use Case in [24].

The following is an example of "building" an XML result from scratch.

TEXT Case - Q2. Find news items where the Foobar Corporation and one or more of its partners are mentioned in the same paragraph and/or title. List each news item by its title and date.

The above query is expressed in X-DEVICE as the follows:

```
if C@company(name='Foobar Corporation',partner.partners ∍ P) and
   N@news_item(title:T $ 'Foobar Corporation' & $ P,date:D)
then news_item1(title:T,date:D)
if C@company(name='Foobar Corporation',partner.partners ∍ P) and
   N@news_item(title:T,*.par.content $ 'Foobar Corporation' & $ P,date:D)
then xml_result(news_item1(title:T,date:D))
```

The `&` operator denotes conjunction of attribute-testing conditions. For example, in the first rule the `title` attribute should contain both the string 'Foobar Corporation' and a string `P` that represents a partner of the above company. Also notice that an attribute can be simultaneously tested and unified with a variable (`title:T`).

The DTD for the above query is:

```
<!DOCTYPE news_item1 [
   <!ELEMENT news_item1 (title, date)>
   <!ELEMENT title (#PCDATA)>
<!ELEMENT date (#PCDATA)>        ]>
```

The structure of the `title` and `date` elements is automatically determined by the type of the `T` and `D` variables, respectively. Notice that the result generates a class/element `news_item1`, because the class/element `news_item` already exists.

5. Conclusions and Future Work

In this paper, we have considered how Web documents, in the XML format, can be intelligently queried and managed using the deductive object-oriented database system X-DEVICE. This has been achieved by storing the XML documents into an OODB through automatic mapping of the schema of the XML document (DTD) to an object schema and storing XML elements as database objects. Our approach maps elements either as objects or attributes based on the complexity of the elements of the DTD, without loosing the relative order of elements in the original document.

Furthermore, X-DEVICE features a deductive rule query language for expressing queries over the stored XML data. The deductive rule language has certain constructs (such as second-order variables, generalized path expressions and ordering expressions) for traversing tree-structured data that were implemented by translating them into first-order deductive rules. The translation scheme is mainly based on the querying of meta-data (meta-classes) about database objects. Comparing X-DEVICE with other XML query languages (e.g. XQuery) seems that the high-level, declarative syntax of X-DEVICE allows users to express everything that XQuery can express, in a more compact and comprehensible way, with the powerful addition of fixpoint recursion and second-order variables.

Users can also express complex XML document views using X-DEVICE, a fact that can greatly facilitate customizing information for e-commerce and/or e-learning [7]. Furthermore, the X-DEVICE system offers an inference engine that supports multiple knowledge representation formalisms (deductive, production, active rules, as well as structured objects), which can play an important role as an infrastructure for the impending Semantic Web. Production rules can also be used for updating an XML document inside the OODB, a feature not yet touched upon the XQuery initiative. However, the study of using production rules for updating XML documents is outside the scope of this paper and is a topic of future research.

Among our plans for further developing X-DEVICE is the definition of an XML-compliant syntax for the rule/query language based on the upcoming RuleML initiative [8]. Furthermore, we plan to extend the current mapping scheme to documents that comply with XML Schema.

6. References

[1] Abiteboul S., Cluet S., Christophides V., Milo T., Moerkotte G., Siméon J., Querying Documents in Object Databases, Int. J. on Digital Libraries, 1(1): 5-19 (1997)

[2] Abiteboul S., Quass D., McHugh J., Widom J., and Wiener J.L., "The Lorel Query Language for Semistructured Data," *Int. Journal on Digital Libraries*, 1(1), pp. 68-88, 1997.

[3] Bassiliades N. and Vlahavas I., "Processing Production Rules in DEVICE, an Active Knowledge Base System", *Data & Knowledge Engineering*, Vol. 24(2), pp. 117-155, 1997.

[4] Bassiliades N., Vlahavas I., and Elmagarmid A.K., "E-DEVICE: An extensible active knowledge base system with multiple rule type support", *IEEE TKDE*, 12(5), pp. 824-844, 2000.

[5] Bassiliades N., Vlahavas I., Elmagarmid A.K., and Houstis E.N., "InterBaseKB: Integrating a Knowledge Base System with a Multidatabase System for Data Warehousing," *IEEE TKDE*, (to appear) 2001.

[6] Bassiliades N., Vlahavas I., Sampson D., Using Logic for Querying XML Data, submitted for publication.

[7] Bassiliades N., Kokkoras F., Vlahavas I., Sampson D., "An Intelligent Educational Metadata Repository to appear in *Intelligent Systems, Techniques and Applications*, C.T. Leondes (ed.), CRC Press, 2002.

[8] Boley H., Tabet S. and Wagner G., Design Rationale of RuleML: A Markup Language for Semantic Web Rules, Int. Semantic Web Working Symposium, Stanford, CA, USA, July/August 2001, pp. 381-402.

[9] Chamberlin D., Robie J., and Florescu D., "Quilt: an XML Query Language for Heterogeneous Data Sources," *Int. Workshop WebDB*, pp. 53-62, 2000.

[10] Chung T.-S., Park S., Han S.-Y., and Kim H.-J., Extracting Object-Oriented Database Schemas from XML DTDs Using Inheritance, *Proc. 2nd Int. Conf. EC-Web 2001*, Munich, Germany, 2001, LNCS 2115, pp. 49-59.

[11] Deutsch A., Fernandez M., Florescu D., Levy A., and Suciu D., "A Query Language for XML," *WWW8 / Computer Networks*, 31(11-16), pp. 1155-1169, 1999.

[12] Deutsch A., Fernandez M.F., Suciu D., "Storing Semistructured Data with STORED," *ACM SIGMOD Conf.*, pp. 431-442, 1999.

[13] Diaz O., Jaime A., "EXACT: An Extensible Approach to Active Object-Oriented Databases", *VLDB Journal*, 6(4), pp. 282-295, 1997.

[14] Florescu D. and Kossmann D., “Storing and Querying XML Data using an RDMBS,” *IEEE Data Eng. Bulletin*, 22(3), pp. 27-34, 1999.

[15] Goldman R., Widom J., DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, *Proc. Int. Conf. VLDB*, 1997, pp. 436-445.

[16] Gray P.M.D., Kulkarni K.G., and Paton N.W., *Object-Oriented Databases, A Semantic Data Model Approach*, Prentice Hall, London, 1992.

[17] Lakshmanan L.V.S., Sadri F., Subramanian I. N., A Declarative Language for Querying and Restructuring the WEB. RIDE-NDS 1996: 12-21

[18] Ludäscher B., Himmeröder R., Lausen G., May W., Christian Schlepphorst, Managing Semistructured Data with FLORID: A Deductive Object-Oriented Perspective, *Information Systems*, Vol. 23, No 8, 1998, 589-613.

[19] May W.: XPathLog: A Declarative, Native XML Data Manipulation Language. IDEAS 2001: 123-128

[20] McHugh J., Abiteboul S., Goldman R., Quass D., Widom J., Lore: A Database Management System for Semistructured Data, *ACM SIGMOD Record*, 26(3),pp. 54-66, 1997.

[21] Naughton J., et al, The Niagara Internet Query System, *IEEE Data Eng. Bulletin*, 24(2), June 2001, 27-33.

[22] Renner A., "XML data and object databases: A perfect couple?", *Proc. Int. Conf. on Data Eng.*, 143-148, 2001.

[23] Shanmugasundaram J., Tufte K., Zhang C., He G., DeWitt D.J., and Naughton J.F., “Relational Databases for Querying XML Documents: Limitations and Opportunities,” *Int Conf. VLDB*, pp. 302-314, 1999.

[24] XML Query Working Group, http://www.w3.org/XML/Query

[25] Yeh C.-L., “A Logic Programming Approach to Supporting the Entries of XML Documents in an Object Database,” *Int. Workshop PADL*, pp. 278-292, 2000.

Appendix A. X-DEVICE Object Schema for the "TEXT" DTD

```
xml_seq company
   attributes        name           (string,single,mandatory)
                     ticker_symbol  (string,single,optional)
                     description    (string,single,optional)
                     business_code  (string,single,mandatory)
                     partners       (partners,single,optional)
                     competitors    (competitors,single,optional)
   meta_attributes   elem_ord       [name,ticker_symbol,description,business_code,partners,
                                    competitors]
xml_seq partners
   attributes        partner        (string,list,mandatory)
   meta_attributes   elem_ord       [partner]
xml_seq competitors
   attributes        competitor     (string,list,mandatory)
   meta_attributes   elem_ord       [competitor]
xml_seq news
   attributes        news_item      (news_item,list,optional)
   meta_attributes   elem_ord       [optional]
xml_seq news_item
   attributes        title          (string,single,mandatory)
                     content        (content,single,mandatory)
                     date           (string,single,mandatory)
                     author         (string,single,optional)
                     news_agent     (string,single,mandatory)
   meta_attributes   elem_ord       [title,content,date,author,news_agent]
xml_seq content
   attributes        content_alt1   (content_alt1,list,mandatory)
   meta_attributes   elem_ord       [content_alt1]
                     alias          [par-content_alt1,figure-content_alt1]
xml_alt content_alt1
   attributes        par            (par,single,optional)
                     figure         (figure,single,optional)
xml_seq par
   attributes        par_alt1       (par_alt1,list,optional)
   meta_attributes   elem_ord       [par_alt1]
                     alias          ['$content'-par_alt1,quote-par_alt1,footnote-par_alt1]
xml_alt par_alt1
   attributes        '$content'     (string,single,optional)
                     quote          (string,single,optional)
                     footnote       (string,single,optional)
xml_seq figure
   attributes        title          (string,single,mandatory)
                     image          (image,single,mandatory)
   meta_attributes   elem_ord       [title,image]
xml_seq image
   attributes        source         (string, single, mandatory)
   meta_attributes   elem_ord       []
                     att_lst        [source]
```