Authors:	Näppilä Turkka, Järvelin Kalervo, Niemi Timo
Name of article:	A Tool for Data Cube Construction from Structurally Heterogeneous XML Documents
Year of publication:	2008
Name of journal:	Journal of the American Society for Information Science and Technology
Volume:	59
Number of issue:	3
Pages:	435 - 449
ISSN:	1532-2882
Discipline:	Natural sciences / Computer and information sciences
Language:	en
School/Other Unit:	School of Information Sciences

URN: http://urn.fi/urn:nbn:uta-3-755
DOI: http://dx.doi.org/10.1002/asi.20756

A Tool for Data Cube Construction from Structurally Heterogeneous XML Documents

Turkka Näppilä,[a] Kalervo Järvelin,[b] and Timo Niemi[c]

Department of Computer Sciences[a,c]; Department of Information Studies[b]

University of Tampere, Finland

E-mail: {Turkka.Nappila; Kalervo.Jarvelin; Timo.Niemi}@uta.fi

Abstract

Data cubes for OLAP (Online Analytical Processing) often need to be constructed from data located in several distributed and autonomous information sources. Such a data integration process is challenging due to semantic, syntactic, and structural heterogeneity among the data. While XML (Extensible Markup Language) is the de facto standard for data exchange, the three types of heterogeneity remain. Moreover, popular path-oriented XML query languages, such as XQuery, require the user to know in much detail the structure of the documents to be processed and are, thus, effectively impractical in many real-world data integration tasks.

Several Lowest Common Ancestor (LCA) -based XML query evaluation strategies have recently been introduced to provide a more structure-independent way to access XML

documents. We shall, however, show that this approach leads in context of certain—not uncommon—types of XML documents to undesirable results. This paper introduces a novel high-level data extraction primitive that utilizes the purpose-built Smallest Possible Context (SPC) query evaluation strategy. We demonstrate, through a system prototype for OLAP data cube construction and a sample application in informetrics, that our approach has real advantages in data integration.

INTRODUCTION

Background

For proper operations, advanced organizations need to analyze both their internal data and data produced by external organizations, such as competitors. This requires data extraction from several autonomous information sources. OLAP (Online Analytical Processing; Codd, Codd & Salley, 1993) is a popular means for analyzing multidimensionally organized summary data for ad hoc information needs. In multidimensional analysis, the underlying summary data are viewed simultaneously along multiple dimensions. Data from operational information systems are aggregated in OLAP into *data cubes* which consist of *dimension* and *measure attributes*. The former provide the factors (e.g., time) along which the values of the latter (e.g., performance, productivity) are analyzed. Individual dimensions may be organized by hierarchical levels of granularity like day–month–year. *OLAP operations*, such as roll-up and drill-down, in association with aggregation functions (e.g., sum, average) afford the opportunity to analyze measure attributes at different levels of granularity to identify interesting changes, prevailing trends, or to compare objects of analysis in the data with each other.

XML (Extensible Mark-up Language; Bray et al., 2004) has become the de facto standard for data representation by removing one of the traditional obstacles of large-scale data exchange —syntactic data incompatibility. Thus, XML offers a natural starting point for data extraction from autonomous and heterogeneous data sources, especially in the Web environment. In XML documents, the logical structures are represented as *elements.* An element is an entity delimited by its *start* and *end tags*, which carry its name. A start tag may also contain *attributes* describing properties or characteristics of an element or carrying an independent data value. An element can consist of nested elements or contain only data values. The user is able to freely name the elements and attributes in his/her XML documents. Their names should be selected carefully, since the semantic interpretation of them is based only on these names. Due to the *semistructured* nature of the XML data, description (schema) and content components are mixed in a document.

Because of the element nesting, XML data are modeled as ordered, labeled trees. This is also the starting point in path-oriented XML query languages developed by the World Wide Web Consortium, the XML standardization organization. These query languages, most notably XQuery, are based on the *path expressions* of the XPath standard (Clark & DeRose, 1999). XPath offers mechanisms both for navigating XML structures and extracting content. An XPath expression specifies the path from the root node of a document to the node of interest. In XPath, elements and attributes are treated differently. XQuery (Boag et al., 2007) has become the most popular XML query language. Besides being a query language, XQuery is also a full-scale, Turing-complete programming language. XQuery has inherited several features from the XPath and XML Schema (Fallside & Walmsley, 2004; Thompson et al., 2004; Biron & Malhotra, 2004) standards. Informa-

tion extraction is done in XQuery by specifying iterative structures and functions. Furthermore, XQuery follows its own, non-XML syntax.

In XML, it is possible to represent semantically similar information in multiple different ways within one document and between documents. This leads to data heterogeneity. XML documents suffer from three types of heterogeneity. In *semantic heterogeneity*, semantically similar information is represented by different names or dissimilar information by the same names. In *syntactic content heterogeneity*, semantically the same content is expressed in different languages (French, English) or units of measurement ($, €, ¥; ^{o}F, ^{o}C). Finally, in *structural heterogeneity* the same or similar data is organized in structurally different ways, e.g., in different levels of hierarchy. In addition to those, a specific piece of information can be represented in XML documents as a name of an element, as a name of an attribute, or as their values. These types of heterogeneity are independent of each other and all combinations among them may appear simultaneously. The heterogeneity between information sources must be harmonized before meaningful data cubes can be constructed based on XML documents. In this paper, we focus on construction of data cubes in structurally heterogeneous XML environments. Thus, we do not consider multidimensional analysis or OLAP per se.

Contribution and Outline

Before the actual OLAP data cube construction, it is necessary to carry out laborious integration among heterogeneous XML data. Popular XML query languages, such as XQuery, are path-oriented and require the user to know the structure of the documents to be processed in much detail. We demonstrate in the present paper that such XML query languages are indeed laborious and troublesome in data integration because the data inte-

grator must explicitly account for structural variety in the documents while (s)he may not be aware of its scope. To relieve the data integrator from this burden, we propose a high-level XML data extraction primitive, which avoids explicit path specification when extracting structurally heterogeneous data from XML-based information sources. We also provide a novel data cube construction operation, which allows high-level declarative expressions for specifying the dimension and measure attributes with their extraction conditions.

In XML data extraction for data cubes, it is necessary to identify which XML components in heterogeneous structures are meaningfully related to each other. The *Lowest Common Ancestor* (LCA) semantics has been proposed to solve the problem in XML query languages (Liu, Yu & Jagadish, 2004; Xu & Papakonstantinou, 2005; Hristidis et al., 2006). We will point out potential shortcomings of the *Smallest Lowest Common Ancestor* (SLCA) semantics (Xu & Papakonstantinou, 2005)—as well as of the similar *Meaningful Lowest Common Ancestor* (Liu, Yu & Jagadish, 2004) and *Minimum Connected Tree* (Hristidis et al., 2006) approaches—which introduce a useful restriction to the LCA semantics and contribute our *Smallest Possible Context* (SPC) evaluation strategy to extract meaningfully related data.

By using *informetrics* (see, e.g., Egghe, 2006) as a sample application, the paper aims to demonstrate that the proposed novel operations and the SPC query evaluation strategy provide a real improvement over path-oriented and LCA-based XML query languages for data integration from structurally heterogeneous XML documents. We assume here that the XML element and attribute names have unambiguous semantics which the

data integrator knows while the data may be organized in heterogeneous structures often largely unknown to him/her.

The rest of the paper is organized as follows. The next section reviews related work on XML-based data cube construction. In the third section, we discuss the requirements for data integration from heterogeneous XML documents and give the goals of our contribution and our heterogeneous XML document collection for the example in informetrics. Thereafter, we shall analyze the integration of XML data and present our data extraction primitive is_component_of and discuss the SPC query evaluation strategy. The following section presents query processing and demonstrates, through sample queries, the benefits of the proposed approach. The paper ends in a discussion section and conclusions.

RELATED WORK

Jensen et al. (2001) present an architecture for integrating XML and relational resources. The architecture contains a component that transforms XML queries into SQL. The mapping between XML and SQL is based on a specific UML diagram, called 'UML snowflake diagram'. Unlike in our approach, the construction of a UML snowflake diagram presupposes that the designer has detail knowledge about the domain. In addition, they focus on finding the multidimensional structures directly in the XML data that are distributed based on, e.g., legal issues or the nature of the data.

Niemi et al. (2002), by contrast, assume that the XML data at hand are intended to be used in multidimensional analysis and are distributed mainly for technical reasons. They present a system where XML is used for data collection to resolve the possible syntactic heterogeneity in data sources. Contrary to our goals, Niemi et al. concentrate on the tech-

nical side of the distribution architecture and they offer no explicit mechanisms for manipulating XML-based information.

Beyer et al. (2005) recognize the limited capabilities for data grouping in XQuery. However, queries requiring grouping of the data are essential in business analytics. Therefore, they propose a new construct that simplifies the query result grouping and provide OLAP-style functionality (aggregation and roll-ups) in XQuery. Further, Wiwatwattana et al. (2007) extend XQuery with an X ^3 operation that allows the manipulation of such XML structures which differ slightly from each other. In our approach, which is not based on XQuery, the SPC query evaluation strategy is utilized to resolve complex heterogeneity among XML structures without the need for user control.

Park et al. (2005) propose a framework for constructing XML data cubes from well-organized XML documents, whereas in our approach the XML documents are assumed to be very heterogeneous. In their approach, an XML data cube is constructed and queried using the XML-MDX language developed for this purpose.

GOALS AND SAMPLE ENVIRONMENT

Requirements for Data Integration with XML

In spite of the data self-description, XML does not specify, (a) should a specific piece of information be represented as an element, attribute or a value; (b) how to label elements and/or attributes; (c) how to structurally arrange the relationships between elements and/or attributes; and (d) how to syntactically represent data values. Feature (b) is an example on semantic heterogeneity and feature (d) on syntactic one. Features (a) and (c) are related to structural heterogeneity, which is our focus.

Due to the above types of heterogeneity, data integration for data cubes is a very demanding task. Therefore, a full-scale system for data cube construction from heterogeneous XML documents should meet the following requirements:

1. *Semantic heterogeneity:* the system should support the identification of semantically equivalent elements and attributes even if their names are inconsistent. Analogously, it should support the automatic integration of such semantically equivalent but inconsistent XML structures.
2. *Syntactic heterogeneity:* the system should support the identification of syntactically equivalent element and attribute values even if their representations (value domains) differ. It should also support automatic harmonization of XML structures.
3. *Structural heterogeneity:* the system should
   - 3.1 not require its user to master the structural diversity in XML structures in detail or to know which kinds of components (elements or attributes) are used to represent the information;
   - 3.2 not require its user to specify explicitly the navigation in XML structures;
   - 3.3 relieve its user from writing complex structural data integration specifications; and
   - 3.4 therefore, execute automatically structural data integration on the basis of compact, declarative and high-level specifications.

In this paper we shall focus on requirements 3.1 to 3.4.

Study Goals

The goal of our study is to support complex data integration tasks for construction of data cubes under the requirements given above. The specific goals are to:

1. demonstrate that popular XPath-based XML query languages, being path-oriented, require the data integrator to know the structure of the documents in too much detail;
2. demonstrate that such XML query languages are too laborious to use in data integration due to the possibility of great structural variation among data of interest;
3. demonstrate that the Lowest Common Ancestor (LCA) semantics, relieving the requirements on structural knowledge in XML query languages, is insufficient;
4. propose a high-level data extraction primitive is_component_of for structural data integration, which avoids explicit path specification and incorporates the novel Smallest Possible Context (SPC) evaluation strategy to extract meaningfully related data from XML documents;
5. introduce a high-level CREATE CUBE operation for data cube construction based on the is_component_of primitive and the SPC query evaluation strategy; and
6. present a sample application in informetrics and sample queries that demonstrate the applicability and salient features of our approach.

Niemi, Hirvonen and Järvelin (2003) introduced an advanced tool for multidimensional analysis based on data cubes. Here we introduce a system through which the user is able to flexibly construct data cubes from available XML documents for such analysis. In our Prolog-based implementation the user constructs a data cube by specifying the CREATE CUBE operation. The operation and the underlying system are described later on. Now, we introduce our structurally heterogeneous sample XML document collection from which sample data cubes for informetrics are later constructed.

A Heterogeneous Sample XML document Collection

Our application area is *informetrics*. Informetrics studies various statistical phenomena of literature which are typically based on bibliographic information provided by online databases. These statistical phenomena may concern productivity issues (of authors, countries, journals; Almind & Ingwersen, 1997), generalized impact factors (of journals, authors; Hjortgaard Christensen, Ingwersen & Wormell, 1997), activity profiles (of authors, organizations), citation networks (bibliographic coupling, author co-citations), literature growth and aging, to mention a few (for more, see, e.g., Egghe, 2006).

Järvelin, Ingwersen and Niemi (2000) presented an easy-to-use interface for generalized informetric analysis. Later, Niemi, Hirvonen and Järvelin (2003) demonstrated that OLAP is a promising approach for advanced analysis in informetrics. However, the prerequisite for OLAP is that there is some underlying data cube, on which the actual analysis is based. We focus on constructing OLAP data cubes based on information extracted from heterogeneous XML documents.

Figure 1. Examples of structural organizations in sample documents of type publications

The sample document collection consists of XML documents of two types, publications and grants. Figures 1 and 2 describe several structural alternatives in sample documents based on the same content. The sample documents of the type publications contain information on publications. Possible element and attribute names used in the documents of this type as well as their possible structures are depicted in Figure 1. We assume that the semantics of these documents are self-explanatory. The information within these documents can be grouped by individual publications (articles or brochures) or according to the *publisher*, *author*, *year*, *domain*, or *type* of an article. Likewise, the sample documents of the type grants contain information on grants. Possible element and attribute names and their structures are given in Figure 2. In this case, information can be grouped by individual grants or by the *institution*, *year*, *grantee*, *domain*, or *amount* of a grant.

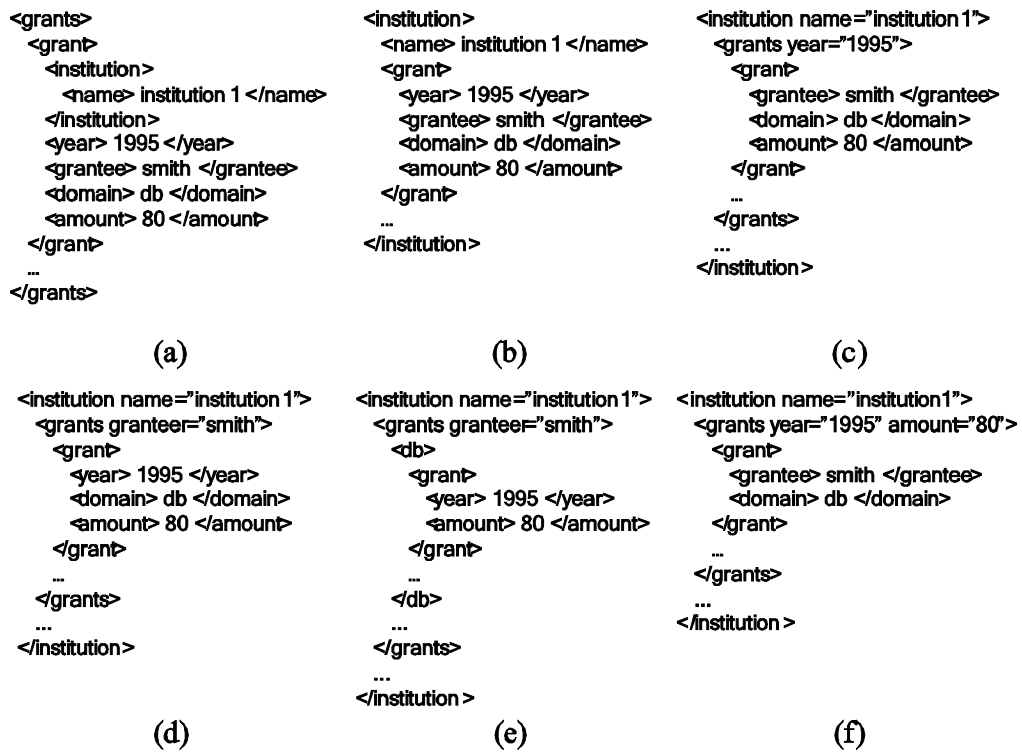

Figure 2. Examples of structural organizations in sample documents of type grants

As Figures 1 and 2 show, the same information within the sample documents can be represented as a value of an element or attribute. Moreover, information about the domain of an article or grant (db or ir) and the type of an article (refereed or non_refereed) can also be expressed by a name of an element. Due to space limitations, the contents of all the sample documents (brochures omitted) are summarized as tables in Appendix A.

INTEGRATION OF XML DATA

The Data Extraction Primitive

We motivate developing a new data extraction primitive by requiring it to be able (1) to handle the elements and attributes of an XML document equally and (2) to utilize both schema and instance level information in a query. First, we argue that in data integration there is no semantic difference between elements and attributes in an XML document

other than that only elements may have substructures. In contrast, popular XML query languages separate elements and attributes. Second, we note that both XML and relational databases share structural heterogeneity problems (see, e.g., Lakshmanan, Sadri & Subramanian, 2001), but unfortunately most relational and XML query languages offer only limited capabilities to utilize schema level information in queries. In XML-based data integration this is, however, often a necessity. Next we take a look at the is_component_of data extraction primitive.

The novel is_component_of primitive is developed for querying the components of an XML document which are its elements and attributes. Generally, a component in an XML document has a name (i.e., the name of an element or attribute) and a value (i.e., the textual linearization of element content or an attribute value). The components of an XML document are specified in the is_component_of primitive with *component expressions*. The primitive has the following basic form:

<*CompExpr2*> is_component_of <*CompExpr1*> in <*DocName*>

where <*Comp_Expr1*> and <*Comp_Expr2*> are component expressions and <*DocName*> is any valid XML document name or a variable referring to an unknown XML document in a document collection.

A component expression has the form <*Name*>(<*Value*>), where <*Name*> is an atom referring to the name of a known component or a variable referring to an unknown component name. If <*Value*> is an atom it expresses explicitly the value of the component <*Name*> whereas as a variable it refers to an unknown value of the component <*Name*>. Variable, as in deductive databases (see, e.g., Liu, 1999), begin with an upper case letter. Strings are written between quotes. The notion of shared variable, typical of logic pro-

gramming (see, e.g., Sterling & Shapiro, 1994) and deductive databases, is used throughout our approach: that is, if several component expressions in a query contain the same variable then its instantiations must be the same throughout the query processing. The <*Name*> is the only compulsory part of a component expression. Note that a component expression matches both elements and attributes without the user needing to know which one it is in the document.

Variables in component expressions can refer to extensional, intensional, or both levels. For example, in the component expression N(smith) the variable N refers to a component with the value smith. Thus, it is related to the intensional level. In the component expression author(V) the variable V is related to the extensional level since it will be instantiated with a value of the component name author. In the component expression N(V), the variable N is related to the intensional whereas the variable V is related to the extensional level. A component expression with two atomic values, for example, author(smith), expresses an explicit condition which, depending on the XML document at hand, may be true or false. A single atom or variable, for example, author or N, is a component expression referring to a known or unknown component name.

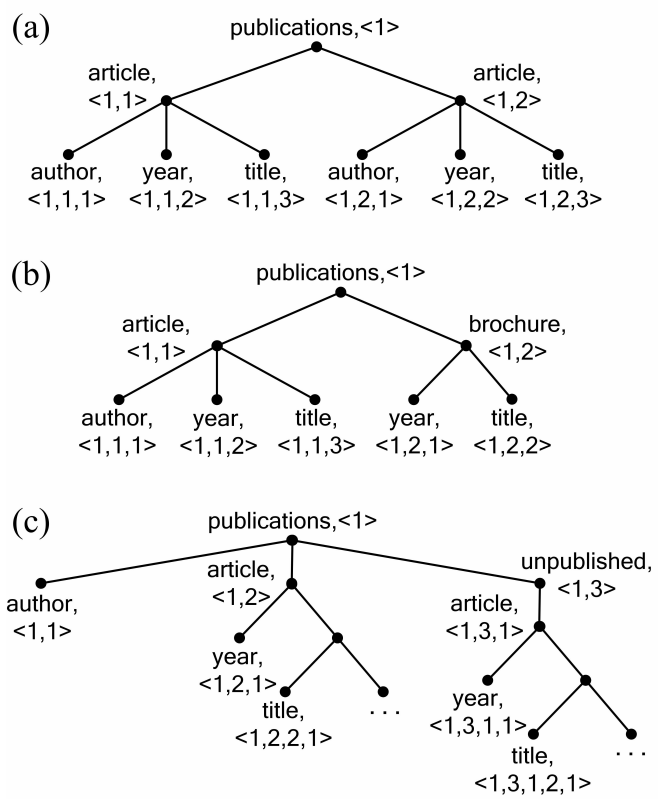

Figure 3. Three examples of data organization in XML documents

The is_component_of primitive may be used in several ways. For example, the expression author(V) is_component_of publications in "example.xml" returns all the components (elements and attributes) with the name author which are, immediate or indirect, subcomponents of the component publications in the XML document example.xml. The corresponding expression in XPath would require the two expressions doc("example")//publications//author and doc("example")//publications//@author. Analogously, the expressions C(V) is_component_of E in D refers to any name (C) and its value (V) of each subcomponent of any element (E) in any XML document (D) in the XML document collection. It is not possible to specify a single XPath expression corresponding to the above is_component_of expression because it presupposes the manipulation of the unknown intensional level.

Lowest Common Ancestor Semantics

Compared to XQuery, the most obvious advantage of the is_component_of query primitive is its capability to query multiple components in a single query expression. In this case, the component expressions are written within braces ({,}) and separated by commas. For example, the expression {author(A), year(Y), title(T)} is_component_of publications in "example.xml" returns all the values of subcomponents author, year, and title associated with the same publications component (publications component may have several occurrences) in the XML document example.xml. The components within the braces may appear in any order and at any nesting levels within their common ancestor element.

However, this approach as such contains one considerable disadvantage. In the cases where the ancestor element contains multiple subcomponents with the same name, all the possible combinations of the occurrences of these subcomponents will be returned. For example, when evaluating the expression above against the XML document depicted in Figure 3 (a) eight sets of component combinations are returned, although it is easy to see that only two of them are meaningful. All other combinations mix information from unrelated components. This is why one has to develop more advanced means to select only the meaningfully related components. For this purpose, several *Lowest Common Ancestor* (LCA) -based semantics have been proposed (Liu, Yu & Jagadish, 2004; Xu & Papakonstantinou, 2005; Hristidis et al., 2006).

The central idea in the LCA semantics is that the given nodes in an XML document are searched only in the context of their *lowest common ancestor* node, thus reducing the search space. To illustrate this, let us first assume that XML documents are indexed with the *structural indices* introduced in (Niemi, 1983). Thus, we denote the root node of an

XML document by the index $<1>$. Any other node in the document is denoted by the index $<\xi,i>$ where ξ is the index of its parent and i an integer that is gained by traversing the node ξ in preorder. For example, in the XML document tree in Figure 3 (a) the LCAs for nodes *author*, *year* and *title* are the nodes with indices $<1,1>$, $<1,2>$, and $<1>$. Note that the application of the LCA semantics does not require such indexing but they are used here only for demonstration.

The problem with the LCA semantics is that the root of an XML document is ultimately the LCA of all the other nodes in any XML document and it, thus, allows descendant nodes to be combined in an arbitrary manner. To prevent this, a restriction to the LCA semantics has been recently developed. The *Smallest Lowest Common Ancestor* (SLCA; Xu & Papakonstantinou, 2005) semantics—like the similar approaches in (Liu, Yu & Jagadish, 2004) and (Hristidis et al., 2006)—modifies to the LCA semantics by requiring that the result must not contain any such a node which is the root of a subtree that also contains all the desired nodes. This restriction is not, however, always sufficient. Next we consider two such cases:

1. A document contains partial information, that is, it has an unequal number of nodes we are interested in. When querying, for example, the nodes *author*, *year*, and *title* in an XML document tree corresponding to Figure 3 (b) in all the LCA-based semantics developed so far, the node $(author,<1,1,1>)$ would also be wrongly associated with the nodes $(year,<1,2,1>)$ and $(title,<1,2,2>)$.
2. A piece of information at a higher level of the document tree must be replicated with the information represented at a lower level of the same tree structure. For example, when querying the nodes *author*, *year*, and *title* in an XML document tree in Figure 3

(c) the node $(author, <1,1>)$ has to be replicated with the right combinations of the nodes *year* and *title*. In LCA -based semantics the *year* and *title* nodes in the document would be combined in an arbitrary manner (e.g., the node $(year, <1,2,1>)$ would be associated with the node $(title, <1,3,1,2,1>)$ because their LCA and SLCA is the node $(publications, <1>)$, i.e., the root of the XML document tree).

The two examples above demonstrate the general problem with LCA -based semantics: they do not offer any mechanism to prevent combining unrelated nodes when some nodes are associated through the root node of the document tree (or its subtree). We have developed the *Smallest Possible Context* (SPC) query evaluation strategy to produce only such node combinations that do not mix data that are not meaningfully related to each other.

Smallest Possible Context Evaluation Strategy

A query evaluated with the SPC query evaluation strategy returns the correct node combinations in all those cases when a query evaluated with some LCA-based semantics does so. In addition, it also returns the correct node combinations in the cases discussed above. The SPC query evaluation strategy does not offer reduced contexts for query evaluation, as LCA-based semantics do, but returns directly a set of meaningfully connected nodes of a given XML document. Unlike the LCA-based semantics, the SPC query evaluation strategy requires indexing of the nodes of XML documents.

Next we demonstrate how meaningfully related nodes are selected in the SPC query evaluation strategy. Let us assume that we want to extract meaningfully related *author*, *year*, and *title* nodes from an XML document organized as in Figure 3 (b). First, we construct the set *A* that contains combinations of any two (*author*, *year*, and *title*) nodes. Each node in a node pair is also equipped with its index. A node pair is chosen into the

set *A* by the criterion that it is not possible to construct any other pair of the same node types whose constituents would have a longer common prefix. The set *A* constructed from the nodes in Figure 3 (b) is

$$A = \{\langle(author,<1,1,1>),(year,<1,1,2>)\rangle, \langle(author,<1,1,1>),(title,<1,1,3>)\rangle,$$
$$\langle(year,<1,1,2>),(author,<1,1,1>)\rangle, \langle(year,<1,1,2>),(title,<1,1,3>)\rangle,$$
$$\langle(year,<1,2,1>),(title,<1,2,2>)\rangle, \langle(title,<1,1,3>),(author,<1,1,1>)\rangle,$$
$$\langle(title,<1,1,3>),(year,<1,1,2>)\rangle, \langle(title,<1,2,2>),(year,<1,2,1>)\rangle\}.$$

The pairs $\langle(author,<1,1,1>),(year,<1,2,1>)\rangle$ and $\langle(author,<1,1,1>),(title,<1,2,2>)\rangle$ are not included in the set *A* because their longest common prefix is $<1>$, that is, it is shorter than the longest common prefix of the indices of the node pairs $\langle(author,<1,1,1>),\ (year,<1,1,2>)\rangle$ and $\langle(author,<1,1,1>),\ (title,<1,1,3>)\rangle$, whose length is 2.

The node pairs of the set *A* can be viewed as arcs in a directed graph. Now, based on these arcs we construct the set *G* that contains all possible *complete graphs* they constitute. In a complete graph all the nodes are adjacent, that is, each node is directly connected to every other node in the graph. In our example, the elements of the set *A* constitute only one complete graph. Thus, the set *G* is

$$G = \{\{\langle(author,<1,1,1>),(year,<1,1,2>)\rangle, \langle(year,<1,1,2>),(author,<1,1,1>)\rangle,$$
$$\langle(year,<1,1,2>),(title,<1,1,3>)\rangle, \langle(title,<1,1,3>),(year,<1,1,2>)\rangle,$$
$$\langle(title,<1,1,3>),(author,<1,1,1>)\rangle, \langle(author,<1,1,1>),(title,<1,1,3>)\rangle\}\}.$$

The arcs $\langle(year,<1,2,1>),(title,<1,2,2>)\rangle$ and $\langle(title,<1,2,2>),(year,<1,2,1>)\rangle$ in the set *A* do not constitute complete graph because no *author* node can be reached from either components.

The smallest possible contexts of the nodes we are interested in are framed by selecting the separate nodes from the graphs in the set *G*. In our example, the smallest possible context for nodes *author*, *year*, and *title* is the sequence $\langle(author,<1,1,1>),(year,<1,1,2>),\ (title,<1,1,3>)\rangle$.

The advantage of the SPC query evaluation strategy lies in choosing the node pairs that constitute the complete graphs based on the longest common prefix of their indices instead of their mutual distance. If the nodes were chosen based on their mutual distance then querying nodes *author*, *year*, and *title* in an XML document organized as Figure 3 (c) would not return the smallest possible context $\langle(author,<1,1>),(year,<1,3,1,1>),(title,<1,3,1,2,1>)\rangle$. This is due to the fact that the distance between the nodes $(author,<1,1>)$ and $(year,<1,3,1,1>)$ is longer than the distance between nodes $(author,<1,1>)$ and $(year,<1,2,1>)$. The node $(year,<1,3,1,1>)$ would be discarded and only the node $(year,<1,2,1>)$ would be selected. However, their longest common prefix is $<1>$ and they are, thus, both selected in the SPC query evaluation strategy. It is worth noting that the longest common prefix of two structural indices is the same as the SLCA of the two nodes. We have embedded the SPC query evaluation strategy as a part of the is_component_of data extraction primitive.

Following the original idea of the LCA-based semantics, the SPC semantics offers a powerful tool for treating the structural obscurity in XML documents. It is often the case that the data integrator does not know the structure of the XML documents in detail but

(s)he, however, knows their contents. If the integrator wants to formulate an effective XQuery expression, (s)he must know the structure of the documents used in the query in some detail. By using the is_component_of data extraction primitive, the user only needs to know the names of elements and attributes used in the XML document collection and, based on this information, the primitive is able to extract all meaningfully related components from these XML documents.

DATA CUBE CONSTRUCTION FROM XML DOCUMENTS

The Data Cube Construction Process

In our approach, a data cube is constructed from the XML documents by the data cube construction operation CREATE CUBE, specified by the user. The operation consists of two parts:

CREATE CUBE *<Cube_Specification>*

WHERE *<Conditions_Specification>*

The cube specification part specifies the content of the data cube. The conditions part gives data extraction conditions in terms of is_component_of expressions.

The cube specification part consists of the cube name and a set of dimension and measure attribute specifications. A *dimension attribute specification* has the form dim(<*Name*>,<*Vars*>), where <*Name*> is the name of the dimension attribute and <*Vars*> refers to a variable (or a list of variables) used in the conditions part. A *measure attribute specification* has analogously the form mes(<*Name*>,<*Var*>,<*Func*>), where <*Name*> is the name of the measure attribute and <*Var*> is a variable whose instantiations are used for the calculation of the values of the attribute based on the aggregation function

<*Func*>. In our implementation, the following aggregation functions are available: count, sum, avg, min, and max.

In the conditions part, the user specifies the extraction of the desired data from the XML documents in terms of is_component_of expressions. By using shared variables in the is_component_of expressions and in the cube specification part, the user associates the cube attributes with relevant information extracted from the documents. If an integration case requires the use of several alternative component expressions then the user should write all the different is_component_of expressions separately.

In order to facilitate the specification of a variety of queries, our system is able to automatically generate the needed is_component_of expressions based on the user's descriptions. Such descriptions consist of two sets of component expressions and of one sample is_component_of expression. Through this sample is_component_of expression the user expresses the relationships between the component expressions in these two sets (i.e., which are subcomponents and which are supercomponents) in the actual is_component_of expressions. The generation process will be explained in detail in the context of Sample Queries. The complete BNF syntax of the CREATE CUBE operation is given in Appendix B.

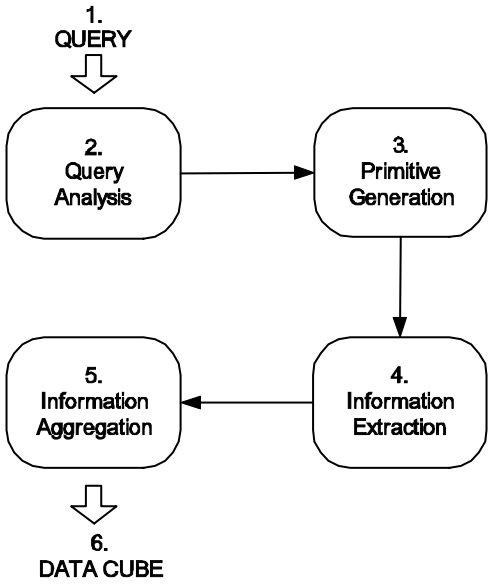

Figure 4. Overview of the system prototype

Before proceeding to the Sample Queries, we briefly look at our system prototype. It has been implemented in Prolog and has a simple text-based interface. An overview of the system implementation is given in Figure 4. First, the user specifies a query (i.e., CREATE CUBE operation) which is parsed by the system. Second, if the query is correct, it will be analyzed for the is_component_of expression generation. Otherwise the user is asked to rewrite the query. Third, a set of is_component_of expressions is generated. Each of the is_component_of expressions is then run in the XML document collection and the extracted information is gathered in the runtime memory as an intermediate result. The intermediate result consists of value sequences whose components are the instantiations of dimension and measure attributes. Fifth, the measure attribute values are aggregated from the intermediate result. The measure attribute values associated with a specific combination of dimension attribute values are gathered and the indicated aggregation function is applied. Finally, the data cube is printed as an XML document.

```
<datacube name="article_cube">
  <tuple>
    <dimension name="year"> 1995 </dimension>
    <dimension name="author"> jones </dimension>
    <dimension name="type"> refereed </dimension>
    <measure name="number_of_articles"> 1 </measure>
  </tuple>
  . . .
</datacube>
```

Figure 5. Sample of the XML representation of a result data cube

A sample of the resulting data cube (the first line of Table C1) is depicted in Figure 5. The root of the XML document is the datacube element. It has one attribute whose value is the data cube name given in the cube specification part. The "rows" of the data cube are represented within tuple elements. The dimension attributes are represented as dimension elements. It has an attribute that expresses its name. The value of the dimension attribute is represented as the value of the element. The measure attributes are analogously represented as measure elements. They contain a name attribute whose value expresses their name. The corresponding data values, based on the calculation given in the cube specification part, are the values of the element.

Sample Queries

Our Sample Query 1 demonstrates a typical multidimensional informetric query by analyzing annual productivity rates of authors. It constructs a data cube that contains the number of articles grouped by the year of publication, by the author, and by the type of an article. The structure of the data cube specification on line (1) is straightforward. The name of the data cube (article_cube), three dimension attributes (year, author, and type) and one measure attribute (number_of_articles) are given. The values of the measure attribute are calculated on the basis of the count of the articles.

Sample Query 1

(1) CREATE CUBE article_cube(dim(year,Y), dim(author,A), dim(type,T),
mes(number_of_articles,N,count))

(2) WHERE C = {year(Y), author(A), title(N), type(T), T = refereed,
T = non_refereed}

(3) S = {publications}

(4) C is_component_of S in DOC.

In the conditions part, the user specifies the XML structures based on the values of which the data cube attributes are populated. On lines (2) and (3) two sets of components expressions are specified. The is_component_of expression description on line (4) shows that the component expressions given in the set C on line (2) occur in the left-hand-side whereas the component expressions in the set S on (3) occur in the right-hand-side of the is_component_of expressions to be generated. In other words, the component expressions in the set S offer a reduced context in which the component expressions included in the set C are evaluated. The expressions T = refereed and T = non_refereed in the set C denote that a value of the variable T is bound to the given element or attribute names and are analogous to component expressions refereed and non_refereed. This means that some generated is_component_of expressions deal with the information represented at the intensional level. The variable DOC will be instantiated with some document name in the XML document collection.

Based on the sets C and S the system generates a set of is_component_of expressions for extracting the desired information from the XML documents. The main principle in the generation process is that each variable occurring in the cube specification part must

occur only once in each generated is_component_of expression. In the context of Sample Query 1, this means that the following three is_component_of expressions are generated:

(a) {year(Y), author(A), title(N), type(T)} is_component_of publications in DOC,

(b) {year(Y), author(A), title(N), T = refereed} is_component_of publications in DOC,

(c) {year(Y), author(A), title(N), T = non_refereed} is_component_of publications in DOC.

The above three is_component_of expressions are, in turn, evaluated in the XML document collection and their result sets are gathered as an intermediate result based on which the data cube is constructed after some data cleaning operations have been executed. Due to space limitations, the result of the Sample Query 1 is represented in tabular form (see Table C1 in Appendix C).

As Sample Query 1 shows, the CREATE CUBE operation offers a very declarative tool for constructing data cubes. A data cube is specified in a simple way by naming the cube, its attributes and the aggregation functions used. Likewise, the query primitives for extracting the required information from XML documents are specified simply by giving the relevant component names or data values. After that the system generates the required query primitives and constructs the data cube automatically. Structures in the source document may vary without effecting the query specification. These features facilitate the data integration considerably from the data integrator's perspective.

Path-oriented XML query languages do not offer any straightforward expression for constructing data cubes. For example, by using XQuery one first needs to specify a set of sub-queries for extracting the desired information from the XML documents. After that one needs to specify a new query that reorganizes the results of the previous sub-queries and aggregates the data values in them. This can become very laborious and troublesome,

in particular, if the user does not know the structures of the documents to be integrated in much detail.

On the basis of the single is_component_of expression on line (a) above one is able to extract information from all the XML documents where the information is structured according to the document fragments (a) – (e) in Figure 1 (and from all possible variations of them, too). By using XQuery one would have to write 14 different queries (for all possible element–attribute combinations) to extract the same information. Further, those 14 XQuery queries do not construct any data cube but they only correspond to the conditions part of CREATE CUBE operation. A sample XQuery query that extracts (partly) the same information than the is_component_of primitive on line (a) is given in Figure 6. This XQuery expression obviously presupposes that the user masters a set of predefined functions and iterative thinking and is also able to synchronize variable instantiations between loops.

```
<result> {
  for $doc in collection (www.example.com/samples)
    for $i in $doc//publication//article
    return<tuple>
      <dim name="year"> data($i//year) </dim>
      <dim name="author"> data($i//author) </dim>
      <dim name="type"> data($i//type) </dim>
      <mes name="title"> data($i//title) </mes>
    </tuple>
} </result>
```

Figure 6. An XQuery query corresponding to an is_component_of expression

When the number of alternative component expressions in the conditions part increases, also the number of is_component_of primitives generated from it increases. For example, if all the component expressions in the sets C and S in Sample Query 1 had three alternative forms, then $243\,(3^5)$ is_component_of primitives should be generated. Most of them would probably return no result. However, we think that the overgeneration of is_component_of primitives is only a minor drawback compared to the great expressive power of the CREATE CUBE operation. Also, sometimes two or more generated is_component_of primitives may extract exactly the same information from an XML document. In this case duplicate information is removed.

If the element and attribute names in the XML documents are ambiguous, that is, semantically different objects are denoted with the same name, in some cases some of the generated is_component_of primitives may return erroneous results. This is because they are too inclusive in combining information from too wide a context. However, this is not a problem solely in our system since similar problems also occur in XQuery and in LCA-based query languages if the nodes in XML documents are named inconsistently.

Sample Query 2 demonstrates a more demanding multidimensional informetric analysis. The user analyzes the correlation between the number of published articles and the total sum of grants grouped by year, person, and domain. The required information has to be extracted from both types of our sample documents (publications and grants). Regardless of the differences in their complexity, the cube specification parts in Sample Queries 1 and 2 resemble each other. The only significant deviation between them is the need to specify two measure attributes in Sample Query 2.

Sample Query 2

(1) CREATE CUBE article_grant_cube(dim(year,Y), dim(person,P), dim(domain,D), mes(no_of_articles,N,count), mes(sum_of_grants,M,sum))

(2) WHERE C1 = {year(Y), author(P), title(N), domain(D), D = db, D = ir},

(3) S1 = {publications},

(4) C1 is_component_of S1 in DOC1,

(5) C2 = {year(Y), grantee(P), amount(M), domain(D), D = db, D = ir},

(6) S2 = {grants, institution},

(7) C2 is_component_of S2 in DOC2

Now, in the conditions part one needs to specify two is_component_of expression descriptions. On lines (2) to (3) and (5) to (6), four sets of component expressions are given. Utilizing these sets the system generates 9 is_component_of expressions based on the descriptions on lines (4) and (7), respectively. The first description produces the following three is_component_of expressions:

(a) {year(Y), author(P), title(N), domain(D)} is_component_of publications in DOC1,

(b) {year(Y), author(P), title(N), D = db} is_component_of publications in DOC1,

(c) {year(Y), author(P), title(N), D = ir} is_component_of publications in DOC1.

Analogously, the second is_component_of expression description generates the following six is_component_of expressions:

(d) {year(Y), grantee(P), amount(M), domain(D)} is_component_of grants in DOC2,

(e) {year(Y), grantee(P), amount(M), D = db} is_component_of grants in DOC2,

(f) {year(Y), grantee(P), amount(M), D = ir} is_component_of grants in DOC2,

(g) {year(Y), grantee(P), amount(M), domain(D)} is_component_of institution in DOC2,

(h) {year(Y), grantee(P), amount(M), D = db} is_component_of institution in DOC2,

(i) {year(Y), grantee(P), amount(M), D = ir} is_component_of institution in DOC2.

The use of the shared variables Y, P, and D in the data cube specification and in all the generated is_component_of expressions enables the system to associate relevant XML structures with the correct data cube attributes. A part of the content of the resulting data cube is given in Table C2 in Appendix C.

The cube specification part of the CREATE CUBE operation remains relatively simple regardless of the complexity of the data cube. The complexity of the conditions part depends on the number of the XML document types, where the required information is to be extracted from, but remains still considerably simple compared to the number of XQuery expressions needed. For example, if the user wants to construct a data cube corresponding to that of Sample Query 2 using XQuery (s)he first needs to formulate explicitly two separate sets of queries. The first contains XQuery expressions for extracting information from the documents of type publications and the second from the documents of type grants. As we demonstrated in the context of Sample Query 1, an is_component_of expression often corresponds to multiple XQuery expressions. In the case of Sample Query 2, we have nine generated is_component_of expressions, so it is not difficult to guess that the number of XQuery expressions the user needs to formulate is large. After formulating and running these XQuery queries, the user needs to combine the result documents returned by each query and finally to execute the required aggregation operations for the data in the combined document. In the CREATE CUBE operation, all this is done automatically and it is invisible to the user.

DISCUSSION

XML greatly supports data sharing between organizations by providing a standard data exchange format. In spite of wide adoption of the XML, the problems related to semantic, syntactic and structural heterogeneity remain. For this reason, XML-based data integration for the OLAP data cube construction is very laborious and troublesome. Our contribution is a powerful tool to facilitate the structural integration problems.

Our study goals were stated in the third section. The sample application in informetrics was purposefully designed to represent structural heterogeneity. The goals 1– 2 (requirement of structural knowledge and laboriousness of XML query languages) were discussed in the context of LCA semantics and through sample queries. Sample queries based on the CREATE CUBE operation remained compact compared to the corresponding expressions in XQuery (shown only partially). In addition, queries based on XQuery were complex and required quite detailed structural knowledge on part of the data integrator. Goal 3 (inefficiency of LCA semantics) was demonstrated through an example showing that LCA semantics and its enhancement (SLCA) do not always extract meaningfully related data from an XML document.

For Goal 4, we introduced the high-level is_component_of data extraction primitive. It is capable of extracting named components—elements or attributes—from any XML document without explicit path specifications or knowledge of the component type. Depending on the use of atomic values and variables in the is_component_of primitive, they can refer to extensional, intensional, or both levels. The proposed Smallest Possible Context (SPC) evaluation strategy extends the LCA semantics to extract meaningfully related data from XML documents.

We also proposed the CREATE CUBE operation for data cube construction (Goal 5). It is based on the is_component_of primitive and the SPC query evaluation strategy. The CREATE CUBE operation allows high-level declarative specification of the target data cube. A set of is_component_of data extraction primitives is automatically generated for each CREATE CUBE operation. In fact, these comprise of all structural combinations which are needed in the extraction. Some of the generated expressions do not extract anything from a given set of XML documents if the specific structural combinations do not exist. On one hand, such over-generation consumes some processing time but on the other hand it relieves the integrator from writing long specifications that would require good knowledge on the underlying XML data.

We chose informetrics as our sample application area as it typically contains much heterogeneity in its data sources (Goal 6). The sample queries demonstrate the applicability and salient features of our approach in informetrics—data cubes may be constructed declaratively through compact expressions that do not require explicit navigation and thus support users who are not aware of the detailed structures of autonomously produced XML documents. The sample queries do not demonstrate the full capabilities of our CREATE CUBE data cube construction operation, for example, the range of aggregation primitives supported.

There are some *limitations in the capabilities* of our data cube construction operation. Consistent with our present focus, it handles only structural heterogeneity—both interesting and demanding per se. Still, both semantic and syntactic heterogeneity, also interesting and demanding problems, remain and we will focus on them in later papers. Järvelin & Niemi (1991) describe our early prototype in automatic resolution of syntactic

heterogeneity. Also, in our approach there are some *basic requirements for the users*. In applying the is_component_of data extraction primitive and CREATE CUBE operation, the data integrator needs to know which basic elements and/or attributes appear in the documents to be integrated. In addition, the semantics of those elements and/or attributes need to be unambiguous. Only structural diversity is accounted for. This may consist of diversity in (a) element structures, (b) using attributes vs. elements for representing information, and (c) using elements and/or attributes names vs. their values for representing information. The user also needs to *understand the notion of shared variables* in the entire CREATE CUBE operation as used in logic programming and deductive databases (see, e.g., Sterling & Shapiro, 1994; Liu, 1999). Unlike logic programming and deductive databases, the user need not master mechanisms of variable instantiation. It is sufficient to use variables intuitively at a high abstraction level.

The above are not hard limitations and requirements and we believe that we take here an important a step forward by offering a powerful tool for automatic data cube construction from heterogeneous XML documents. As discussed above, the conventional approaches have more severe limitations and requirements. For example, in XQuery the requirement to know in fairly much detail the structures of the XML data to be integrated can only be met with much effort in the general case. Also the requirement on explicit navigation in XML structures requires, in addition, iterative thinking and synchronization of variable instantiations in nested loop structures. In complex cases, the user has to construct multiple complex XQuery expressions for a single data integration task. Moreover, XQuery does not provide high-level primitives for data cube construction, even if the basic data would be available. We consider the following features of the CREATE CUBE op-

eration in data cube construction most salient: it supports declarative, compact expressions that do not require explicit navigation, and such users who do not know the heterogeneous structures in the source XML documents in detail.

CONCLUSIONS

OLAP (Online Analytical Processing) is a modern means for analyzing multidimensional summary data for ad hoc information needs. It is used to analyze both internal data of organizations and, increasingly, data produced autonomously by external organizations, such as competitors. The integration of autonomously produced data in lack of clear and agreed domain-specific standards leads to semantic, structural and syntactic data heterogeneity. While XML greatly supports data sharing by providing a standard exchange format, the problems of heterogeneity remain. This makes data integration for OLAP data cube construction very laborious.

We argued that popular XML query languages, such as XQuery, require the data integrator to know the structure of the documents to be integrated in much detail. We demonstrated that typical XML query languages are indeed laborious in this task. We also pointed out that the Lowest Common Ancestor (LCA) semantics, proposed to relieve the requirements on structural knowledge in XML query languages, is insufficient. We contribute a novel high-level operation for data integration, which avoids explicit path specification and relieves the user from knowing whether the actual information in XML documents is represented as elements or attributes. We also specified the Smallest Possible Context (SPC) query evaluation strategy as an extension to LCA semantics to easily extract meaningfully related data from XML documents. Finally, we demonstrated,

through a sample application in informetrics, the salient desirable features of our approach to data integration.

ACKNOWLEDGEMENTS

This study was funded in part by the Academy of Finland under grant numbers 1209960 and 204978 (Multilingual and Task-based Information Retrieval), the University of Tampere, and the Tampere Graduate School in Information Science and Engineering (TISE).

REFERENCES

Almind, T. C., & Ingwersen, P. (1997). Informetric analyses on the World Wide Web: Methodological approaches to "webometrics". Journal of Documentation 53(4), 404-426.

Beyer, K., Chambérlin, D., Colby, L.S., Özcan, F., Pirahesh, H., Xu, Y. (2005). Extending XQuery for Analytics. Proceedings of the SIGMOD 2005 (pp. 503-514), Baltimore, MD.

Biron, P. V., & Malhotra, A. (2004). XML Schema Part 2: Datatypes (2nd ed.) (W3C Recommendation). Retrieved August 24, 2006, from http://www.w3.org/TR/xmlschema-2/.

Boag, S., Chamberlin, D., Fernàndes, M., Florescu, D., Robie, J., & Siméon, J. (2007). XQuery 1.0: An XML Query Language (W3C Recommendation). Retrieved January 23, 2007, from http://www.w3.org/TR/xquery.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Eve Maler, E., Yergeau, F., & Cowan, J. (2004). Extensible Markup Language (XML) 1.1: Third Edition (W3C Recommendation). Retrieved August 24, 2006, from http://www.w3.org/TR/xml11.

Clarke, J., & DeRose, S. (1999). XML Path Language (XPath). Version 1.0 (W3C Recommendation). Retrieved August 24, 2006, from http://www.w3.org/TR/xpath.

Codd, E. F., Codd, S. B., & Salley, C. T. (1993). Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate (Technical Report). Sunnyvale, CA: E. F. Codd and Associates, 1993.

Egghe, L. (Ed.). (2006). Informetrics [Special Issue]. Information Processing and Management, 42(6), 1405-1656.

Elmasri, R., & Navathe, S. B. (2006). Fundamentals of database systems (5th ed.). Boston, MA: Pearson / Addison-Wesley.

Fallside, D. C., & Walmsley, P. (2004). XML Schema Part 0: Primer. (2nd ed.) (W3C Recommendation). Retrieved August 24, 2006, from http://www.w3.org/TR/xmlschema-0/.

Hjortgaard Christensen, F.,Ingwersen, P.,& Wormell, I. (1997). Online determination of the journal impact factor and its international properties. Scientometrics, 40(3), 529-540.

Hristidis, V., Koudas, N., Papakonstantinou, Y., & Srivastava D. (2006). Keyword proximity search in XML trees. IEEE Transactions on Knowledge and Data Engineering 18(4), 525-539.

Jensen, M.R., Møller, T.H., & Pedersen, T.B. (2001). Specifying OLAP cubes on XML data. JIIS 17(2-3), 255-280.

Järvelin, K., Ingwersen, P., & Niemi, T (2000). A user-oriented interface for generalised informetric analysis based on applying advanced data modelling techniques. Journal of Documentation 56(3), 250-278.

Järvelin, K. & Niemi, T. (1991). Data conversion, aggregation and deduction in advanced retrieval from heterogeneous fact databases. In A. Bookstein et al. (Eds.), Proceedings of the SIGIR 1991 (pp. 173-182), Chicago, IL. New York, NY: ACM Press.

Lakshmanan, L. V. S., Sadri, F., & Subramanian, S. N. (2001). SchemaSQL: An extension to SQL for multidatabase interoperatibility. TODS 26(4), 476-519.

Liu, M. (1999). Deductive database languages: Problems and solutions. ACM Computing Surveys, 31(3), 27-62.

Liu, Y., Yu, C., & Jagadish, H. V (2004). Schema-free XQuery. Proceedings of the VLDB 2004 (pp. 72-83), Toronto, Canada.

Niemi, T., Niinimäki, M., Nummenmaa, J., & Thanisch, P. (2002). Constructing an OLAP cube from distributed XML data. Proceedings of the DOLAP 2002 (pp. 22-27), McLean, VA.

Niemi, T. (1983). A seven-tuple representation for hierarchical data structures. Information Systems 8(3), 151-157.

Niemi, T., Hirvonen, L., & Järvelin K. (2003). Multidimensional data model and query language for informetrics. JASIST 54(10), 939-951.

Park, B.-K., Han, H., & Song, I.-Y. (2005). XML-OLAP: A multidimensional analysis framework for XML warehouses. In A.M. Tjoa, J. Trujillo (Eds.): Proceedings of the DaWaK 2005 (pp. 32-42), Copenhagen, Denmark, LNCS 3589. Berlin-Heidelberg, Germany: Springer-Verlag.

Sterling, L. & Shapiro, E. (1994). The art of Prolog: Advanced programming techniques (2nd ed.). Cambridge, MA: MIT Press.

Thompson, H. S., Beech, D., Maloney, M., & Mendelsohn, N. (2004). XML Schema Part 1: Structures (2nd ed.) (W3C Recommendation). Retrieved August 24, 2006, from http://www.w3.org/TR/xmlschema-1/.

Wiwatwattana, N., Jagadish, H. V., Lakshmanan, L.V.S., Srivastava, D. (2007). X^ 3: A cube operator for XML OLAP. Proceedings of the ICDE 2007 (pp. 916-925), Istanbul, Turkey.

Xu, Y. & Papakonstantinou, Y. (2005). Efficient keyword search for smallest LCAs in XML databases. Proceedings of the SIGMOD 2005 (pp. 527-538), Baltimore, MD.

APPENDIX A. CONTENTS OF THE SAMPLE DOCUMENTS

Table A1

Contents of the Sample Documents of the Type publishing

article	author				year	publisher	domain	type
	smith	jones	hikes	wilkins				
art1	x				1995	publisher4	db	refereed
art2	x	x			1995	publisher2	ir	refereed
art3				x	1995	publisher1	db	refereed
art4		x	x		1996	publisher3	db	non_refereed
art5	x				1996	publisher3	ir	refereed
art6		x		x	1997	publisher2	ir	non_refereed
art7	x	x		x	1997	publisher4	db	refereed
art8	x				1997	publisher4	db	refereed
art9	x				1998	publisher1	ir	refereed
art10		x	x	x	1998	publisher2	db	non_refereed
art11	x		x		1999	publisher1	db	refereed
art12		x			1999	publisher1	ir	refereed
art13			x	x	1999	publisher4	db	non_refereed
art14				x	2000	publisher3	ir	refereed
art15	x		x		2000	publisher2	db	refereed
art16	x				2001	publisher3	db	refereed
art17		x			2001	publisher4	db	refereed
art18	x	x	x		2001	publisher2	ir	refereed
art19			x	x	2002	publisher4	ir	non_refereed
art20				x	2002	publisher1	db	refereed
art21	x		x		2003	publisher1	db	refereed
art22	x			x	2003	publisher3	db	refereed
art23		x	x		2003	publisher4	ir	non_refereed
art24		x		x	2004	publisher3	db	non_refereed
art25	x	x		x	2004	publisher2	ir	refereed

Table A2

The contents of Sample Documents of type grants

year	db							ir							author
	inst1	inst2	inst3	inst4	inst5	inst6	inst7	inst1	inst2	inst3	inst4	inst5	inst6	inst7	
1995	80	25								5				10	smith
1996	20	15												20	
1997	30	5		25	3								25	20	
1998	10				2										
1999	100	25													
2000	100	20													
2001	50	15		25	4										
2002	100									25		4	25	20	
2003	10				4					5		2			
2004	100	25		25	4										
1995	20											3			jones
1996		15										3			
1997	10				3							6		20	
1998	50	20		5	3					15			100	20	
1999	20				3							1	25	20	
2000	30			5						10	25		25		
2001	10				2					10		3		20	
2002		20			4							6			
2003	50				6					25		3	70	20	
2004													100		
1995	50	10													hikes
1996	50											1			
1997															
1998															
1999															
2000	20	10		5								3			
2001	20	15		5						15		2			
2002	20	25		5	4							6			
2003	30	15								15					
2004	50	25		5						10		3			
1995		10										2		10	wilkins
1996	20	15		5						15		2			
1997	30	15		5	4					10	25				
1998	100	15		5	4									20	
1999	25	25		5	3					25	25		25		
2000	20	20		5						25	25	5	25		
2001		10		5	2						25	3			
2002	20				2					5	25	2			
2003	100	15		5						5	25			20	
2004	100	20								25	25		35	25	

APPENDIX B. BNF GRAMMAR FOR THE CREATE CUBE OPERATION

The grammar is given using the extended BNF notation (International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 14977: Information Technology – Syntactic Metalanguage – Extended BNF. International Organization for Standardization, Geneva, Switzerland, 1996).

```
query = create_cube, " ", where ;
create_cube = "CREATE CUBE", " ", cube_specification ;
cube_specification = cube_name, "(", dim, (",", dim)*, ",", mes, (",", mes)*, ")" ;
cube_name = atom ;
dim = "dim", "(", atom, ",", variable, (",", variable)*, ")" ;
mes = "mes", "(", atom, ",", variable, ",", func, ")" ;
func = "min" | "max" | "count" | "sum" | "avg" ;
where = "WHERE", " ", model_expression, (",", model_expression)* ;
model_expression = set_expression, ",", set_expression, ",", ico_expression ;
set_expression = variable, " ", "=", " ", "{", comp_expression, (",", comp_expression)*, "}" ;
comp_expression = atom | atom, "(", atom, ")" | atom, "(", variable, ")" |
                  variable, "(", atom, ")" | variable, "(", variable, ")" |
                  variable, "=", atom ;
ico_expression = variable, " ", "is_component_of", " ", variable, " ", "in", " ", variable ;
atom = lc_letter+, (lc_letter* | uc_letter* | char*)* ;
variable = uc_letter+, (lc_letter* | uc_letter* | char*)* ;
lc_letter = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n"
          | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" ;
uc_letter = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N"
          | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" ;
char = "-" | "_" ;
```

APPENDIX C. RESULTS OF THE SAMPLE QUERIES

Table C1

The Result of Sample Query 1

year	author	type	number_of_articles
1995	jones	refereed	1
1995	smith	refereed	2
1995	wilkins	refereed	1
1996	hikes	non_refereed	1
1996	jones	non_refereed	1
1996	smith	refereed	1
1997	jones	non_refereed	1
1997	jones	refereed	1
1997	smith	refereed	2
1997	wilkins	non_refereed	1
1997	wilkins	refereed	1
1998	hikes	non_refereed	1
1998	jones	non_refereed	1
1998	smith	refereed	1
1998	wilkins	non_refereed	1
1999	hikes	non_refereed	1
1999	hikes	refereed	1
1999	jones	refereed	1
1999	smith	refereed	1
1999	wilkins	non_refereed	1
2000	hikes	refereed	1
2000	smith	refereed	1
2000	wilkins	refereed	1
2001	hikes	refereed	1
2001	jones	refereed	2
2001	smith	refereed	2
2002	hikes	non_refereed	1
2002	wilkins	non_refereed	1
2002	wilkins	refereed	1
2003	hikes	non_refereed	1
2003	hikes	refereed	1
2003	jones	non_refereed	1
2003	smith	refereed	2
2003	wilkins	refereed	1
2004	jones	non_refereed	1
2004	jones	refereed	1
2004	smith	refereed	1
2004	wilkins	non_refereed	1
2004	wilkins	refereed	1

Table C2

The Result of Sample Query 2

year	person	domain	number_of_articles	sum_of_grants
1995	hikes	db	0	60
1995	jones	db	0	20
1995	jones	ir	1	3
1995	smith	db	1	105
1995	smith	ir	1	15
1995	wilkins	db	1	10
1995	wilkins	ir	0	12
1996	hikes	db	1	50
1996	hikes	ir	0	1
1996	jones	db	1	15
1996	jones	ir	0	3
1996	smith	db	0	35
1996	smith	ir	1	20
1996	wilkins	db	0	40
1996	wilkins	ir	0	17
. . .	. . .	. . .	. . .	. . .
2002	hikes	db	0	54
2002	hikes	ir	1	6
2002	jones	db	0	24
2002	jones	ir	0	6
2002	smith	db	0	100
2002	smith	ir	0	74
2002	wilkins	db	1	22
2002	wilkins	ir	1	32
2003	hikes	db	1	45
2003	hikes	ir	1	15
2003	jones	db	0	56
2003	jones	ir	1	118
2003	smith	db	2	14
2003	smith	ir	0	7
2003	wilkins	db	1	120
2003	wilkins	ir	0	50
2004	hikes	db	0	80
2004	hikes	ir	0	13
2004	jones	db	1	0
2004	jones	ir	1	100
2004	smith	db	0	154
2004	smith	ir	1	0
2004	wilkins	db	1	120
2004	wilkins	ir	1	110