PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 50, NO. 3–4, PP. 223–240 (2006)

A MODAL LOGICAL APPROACH FOR DEVELOPING XML DATABASES

Dániel SZEGŐ

Department of Measurement and Information Systems
Budapest University of Technology and Economics
H–1521, pf. 91, Budapest, Hungary
e-mail: szegod@mit.bme.hu

Received: April 28, 2006

Abstract

This paper investigates the possibility of realizing the core of an XML database system by a pure modal logical formalism providing query and constraint languages with well-defined syntax semantics and computational elements. The paper also introduces a domain-specific modal logic for XML documents which can be used to implement some of the basic services of an XML database.

Keywords: XML database, modal logic, description logic.

1. Introduction

During the last few years, several attempts were made to develop databases and database technologies for XML documents. These attempts can be separated into two major categories. On the one hand, existing relational database technology is extended to handle XML documents [1]. These attempts try to exploit the precise theoretical framework of the relational algebra and extend it to Web documents [2]. On the other hand, Xpath (XML Path language) is being extended to an XML query language (XQuery) [3, 4]. However, this direction is in lack of simple formal semantics or efficient well-known algorithms. Unfortunately, no real industrial solution with precise theoretical framework and fast algorithms is known at this point.

Modal, especially description logics are simple logical formalisms which primarily focus on describing terminologies and graph-style knowledge [5]. For example, Entity-Relationship diagrams or semantic networks can easily be translated to description logical formulas, having a more precise representation than the original ones [6]. Hence, these logics have pretty good computational properties, like EXPTIME satisfaction algorithms. Therefore, description logics seem to be adequate basis for developing a common computational environment for several Web document processing tasks.

This paper investigates the possibility of realizing the core of a database system by a pure modal logical formalism providing query and constraint languages with well-defined syntax semantics and computational elements. The paper also

introduces a domain-specific modal logic for XML documents which can be used to implement some of the basic services of an XML database.

The reminder of this paper is organized as follows. Related work is surveyed in Section 2. Section 3 introduces the basic architecture of the database. Physical and Conceptual layers of the database are discussed in Section 4 and Section 5. Section 6 covers the language layer including query and constraint expressions in details. Algorithms of the database engine are briefly introduced in Section 7. Section 8 deals with some implementation issues. Finally, Section 9 draws some conclusions.

2. Related Work

There are several different approaches which try to develop either a query language or query and transformation framework for XML documents [7]. LOREL was originally designed for querying semi-structured data. The original data was extended to handle XML documents as well. It is a user-friendly language in SQL/OQL style, which includes a strong mechanism for type coercion and permits very powerful path expressions [8]. Similarly to XQuery, it is rather a query and transformation framework than a simple query language. XML-GL is a graphical query language relying on graphical representation of XML documents and DTDs. The graphical representation is a special labelled graph. All elements of XML-GL are displayed visually, so it is suitable for supporting user-friendly interfaces [9]. There are some approaches which try to strongly integrate traditional relational database systems with XML querying. In this way, a lot of elements of the database stand ready and can be used without further development. Such elements are query language, transaction or multi-user support. Unfortunately, the biggest problem with these approaches is that traditional database cannot be applied straightforwardly for XML documents. Such approach database system based approaches are XREL [10], XML-QL [11]. There are also some query languages which rather provide an XSLT style old-fashioned template matching language than an Xquery style approach. For example, both TQL and XQL can be regarded as such approaches. Although, they did not reach the real industrial level, but there are ideas in which they exceed XSLT.

XML query languages provide practical approaches of XML querying. However, the field is also studied from theoretical point of view. Of course practical and theoretical considerations and approaches often meet. For example Xquery is an industrial approach, but it has a nice formal semantics which is based on an XML query algebra. There are two major theoretical categories of XML querying. On the one hand, different algebras are used for querying XML documents [12]. These approaches try to make benefits of the ideas of relational database. They try to develop a simple but powerful algebra and solve query problems with such algebras. Most elements of most query algebras are similar to operators of the relational algebras, like projection, selection or joins. Perhaps the most important

algebra is XML Query algebra which provides the basis for the W3C standards query language XQuery. Certainly there are further important approaches for XML algebras, but they can be divided into two major categories. Some approaches develop quite similar algebras for XML Query Algebra, but modify a few of its elements to improve expressive power or efficiency. Such approaches are for example, XAL[13] or TAX [14] which is a dedicated solution for query optimization. On the other hand, there are some solutions which do provide a general query language; instead they focus on some special areas like data mining [15].

Other approaches for XML querying try to extend techniques from handling semi structured data to Web documents [16]. Semi structured data contain data fragments and a relation between data fragments, which relation is usually a partial order [17]. There are some mathematical approaches which deal with the possibilities of realizing constraint or query systems for semi structured data. Unfortunately, this research field is quite theoretical. NP or decidable properties of an approach are usually given, but approaches of the field are in lack of efficient polynomial algorithms. On the other hand, a Web document is not really a simple semi structured data. Of course, embedding of tags of an XML document can be regarded as a partial relation over tags. However, even an XML document contains a lot of other elements, like next relation between tags, attributes, values, names, types or even links. Hence, HTML document is even farer from a simple semi structured data.

Both query algebras and approaches for semi-structured data provide a simple but powerful theoretical environment. Unfortunately, neither filed have produced such solution that covers most or at least some areas of structural Web document processing. Query algebras are good for defining query or transformation languages for Web documents, but the definition of constraints, schemas or search languages have not been really studied yet. Similarly, approaches for semistructured data provide beautiful theoretical approaches but they are not really scaled up to Web document.

The basic motivation behind this work is to develop a common logical approach, called SDL (Structured Document Logic) for several different Web document processing fields [18]. These fields are among the others, XML schemas, XML constraints, XML querying and special search functions. Since SDL is based on modal logic, it has all benefits of the short formal semantics. An XML database seemed to be quite a good test case for SDL, because most of the document processing tasks appear (e.g. search, query, categorization, constraint, database schemas…). A core database means that it contains most basic elements of a database, but it is not an absolute industrial one. The major motivation behind this database is to demonstrate that most XML database elements can be realized by practical and theoretical foundations of SDL. I did not want to develop a full industrial database which can be on the market within a few weeks. Consequently, the major focus was on developing a document representation, database schemas, a query language, a constraint language, DDL (Data Definition Language), DML (Data Modification Language). Little attention was paid on developing multi-user support or programming language independency.

3. Architecture

Fig. 1 demonstrates the basic architecture of the XML database. At the lowest level, called physical level, XML documents are stored in a simple file system. The database does not really effect the lowest level, so documents can be reached and transformed by traditional ways as well without making benefits of the database management system. Conceptual level is the first real layer of the modal database system. Firstly, it consists of the document representations, which are the mathematical formalizations of XML documents stored at the physical level. Secondly, it contains conceptual dependencies between the individual documents, which are manifested as an abstract directory service (from a database theorist point of view one could also call a hierarchical view system). Thirdly, both documents and directories can be validated or invalidated by the constraint system of the database. Actually, constraint system could have been placed both at conceptual and language level (it is expressed by drawing constraints as a standalone component). On the one, it distinguishes between valid and non-valid documents, so it should be regarded as part of the conceptual level. However, its working mechanism is based on a constraint language processed by the database engine, so it is very similar to components of the language level.

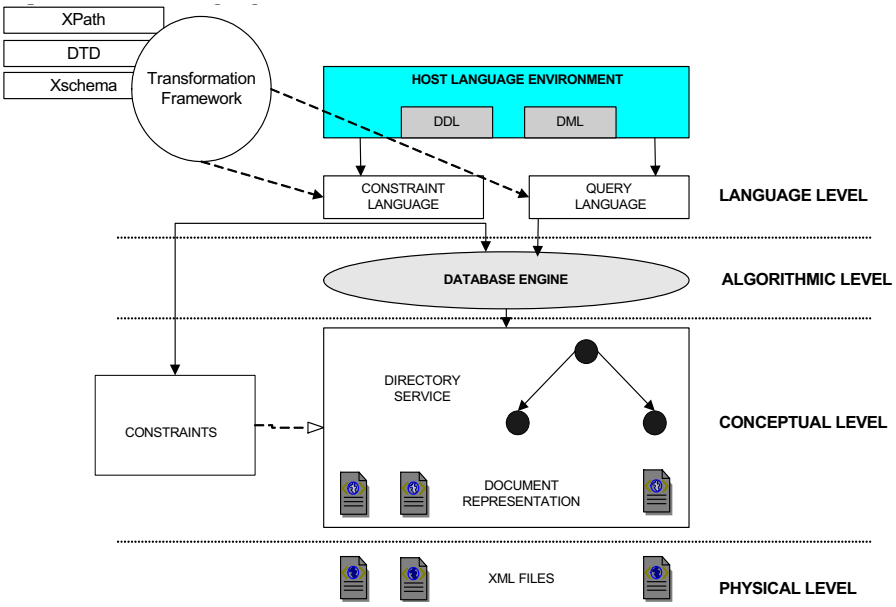

Fig. 1. Basic architecture of the XML database

Perhaps, one of the most important layers is the algorithmic one. It contains all the core algorithms of the database system, in other words the database engine. Since the theory is based on modal logic, the layer contains, among the others,

several different model checking algorithms.

The highest layer of the architecture is the language one. It consists of simple query and constraint languages which are based on modal logic. Most database systems consist of an imperative language beside the query and constraint language. For example, at Oracle databases the standard SQL language is extended by imperative elements called PL/SQL. In our case, there is no such extension. However, the whole language level is embedded into a host language so that imperative programs could be written in conjunction with query or constraint statements. Language level also contains the transformation frameworks of the previous chapter. Consequently, Xpath, DTD and Xschema expressions can be translated to query or constraint level statements.

In the followings, each layer of the architecture will be introduced in details.

4. Physical Level

The physical level supervises the lowest level storage of XML files. Each file has a corresponding formal document model at the conceptual level to realize the document representation. At physical level, there are three principal simple functions to supervise the life cycle of the documents.

- 'load': It loads a new document from the outer file system and creates its formal document model. It can also manage where a document is placed in the database management system.
- 'delete': It eliminates an XML file and all of its document representation.
- 'write-out': Some files are stored only as document representations. The function makes an XML file from the formal document models of these files.

There are two kinds of files at the physical level, the stored and the non-stored files. Stored files have double storage, they are both stored on a file system as XML files and in the database as formal document model. Non-stored files are stored only in the database, so one needs to use 'write-out' to get a real XML file. Outer programs and mechanisms can access stored files. Consequently, each time the database accesses a stored file, the file must be read out and the document representation must be refreshed. In this way, a preliminary version of transaction handling and multi-user support can be achieved with the help of running many entities from the database (technically, transaction and multi-user support of the file system is used instead of developing an absolute new one).

SDL does not really appear at physical level; instead, the level contains lots of lower-level administration tasks.

5. Conceptual Level

This section discusses two major elements of the conceptual level, document model and directory representation, whilst constraints are discussed in the following sec-

tion. Both definitions can be regarded from both practical and theoretical points of view. From practical point of view, document model is the mathematical formalization of an XML document, which is represented as a relational structure (basic set, maps and relations on the basic sets). From theoretical point of view, document model is a model of a logic (similarly to models of the first order logic). The same is true for the directory representation.

The ***formal document model*** is relational structure defined by the following sets maps and relations.

$<N, AN, AV, r, c, a, n, nt, nn, at, an, av, R>$.

- N is a finite set of nodes or set of normal nodes of the structure.
- AN is a finite set of attribute nodes.
- AV is a possible infinite set of atomic values.
- $r \in N$ is the root of the structure.
- $c \subseteq N \times N$ is a binary relation associating each node with its children nodes.
- $a: N \to 2^{AN}$ is a partial map associating each node with a set of attribute nodes.
- $n: N \to N$ is a partial map associating each node with its following (next) node.
- $nt: N \to 2^{AV}$ is a partial map associating each node with a set of type labels.
- $nn: N \to AV$ is a partial map associating each node with a name label.
- $at: AN \to 2^{AV}$ is a partial map associating each attribute node with a set of type labels.
- $an: AN \to AV$ is a partial map associating each attribute node with a name label.
- $av: AN \to AV$ is a partial map associating each attribute node with a value label.
- $R = \{r_1, r_{2\ldots}\}$ is an enumerable set of relations, where each $r_i \subseteq (N \cup AN) \times AV_1 \times \ldots \times AV_{k(i)}$.

Naming conventions:

- $k(i)$ number of r_i relation is called argument number of r_i.
- General edges of the graph are represented as $<x,y>$ elements of c, n or a ($<x,y> \in c$, $n(x)=y$, $y \in a(x)$). One can also distinguish child, next or attribute edges; general edges are the union of the three ones.
- Children paths of the graph are represented by $<n_1, n_2, n_3, \ldots n_{N-1}, n_N>$ sequences, where $n_i \in N$, $<n_i, n_{i+1}> \in c$.

Axioms of the structure are the followings:

1. Children paths of the graph must form a tree, each maximal long children path has to start from the 'r' root node, and each node of the graph must be reached from the top node through one of the children paths.
2. Following nodes must have the same parent: $n(v_i) = v_j$ implies that there exists a $v_k \in N$, for which $<v_k, v_i> \in c$ and $<v_k, v_i> \in c$.
3. Two children nodes of a common parent must be linked by next maps: $<v_k, v_i> \in c$ and $<v_k, v_j> \in c$ implies that there exists a sequence of nodes $<v_1, v_2, v_3, \ldots v_{N-1}, v_N>$ such that $n(v_p) = v_{p+1}$ and $v_1 = v_i, v_N = v_j$ or $v_1 = v_j, v_N = v_i$.

4. Each attribute node is associated with one and only one node: $a(v_1) \cup a(v_2) \cup \ldots \cup a(v_{\#N}) = AN$, and for each $i,j \in \#N\ a(v_i) \cup a(v_j) = \emptyset$.

This definition seems natural for an XML document. For example tags can be translated to nodes and embedding of tags represents the children relation. It is less trivial for an HTML document, because HTML standards are much less strict. Hence there are a lot of pages which do not follow even the HTML standards. Consequently pre-transformations and pre-filters need to be applied. These transformations attempt to capture the necessary parts of an HTML document for a given task. Document model will be set up with these pre-filtered parts. The transformations are strongly task dependent and consist of plenty ad-hoc mechanisms.

Special attribute nodes describe attributes of tags of XML or HTML pages. Each attribute node is connected to a simple node by the '*a*' binary relation. Each attribute node is associated with three further elements: an attribute name, an attribute type and an attribute value. For representing types, both attribute and normal nodes are linked by type symbols.

As a simple example, the following figure demonstrates an HTML fragment, and the corresponding document model.

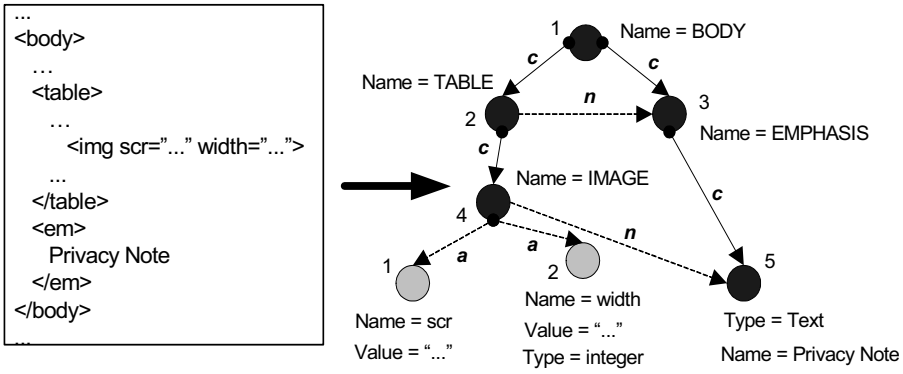

Fig. 2. Example

Black circles of *Fig. 2* represent the standard nodes of the document model (N set), whilst grey nodes are the attribute nodes. For the sake of simplicity, name value and type elements of a node are represented as simple list of values, so '*nt*', '*nn*', '*an*', '*at*' and '*av*' maps do not appear explicitly. '*c*' and '*n*' relation and map appear as links between the standard nodes and '*a*' map is represented by the links between simple and attribute nodes.

The exact formal document model is the following:

N = $\{n_1, n_2, n_3, n_4, n_5\}$.
AN = $\{a_1, a_2\}$.
AV = {"…", BODY, TABLE, EMPHASIS, IMAGE, Text, Privacy Note, width, scr, integer}.
r = $\{n_1\}$.
c = $\{<n_1, n_2>,<n_1, n_3>,<n_2, n_4>,<n_3, n_5>\}$.
n = $\{n(n_2) = n_3, n(n_4) = n_5\}$.
a = $\{a(n_4) = \{a_1, a_2\}\}$.
nt = $\{nt(n_5) = \{Text\}\}$.
nn = $\{nn(n_1)$ = BODY, $nn(n)_2) = TABLE, nn(n_3) = EMPHASIS, nn(n_4) = IMAGE, nn(n_5) = Privacy\ Note\}$.
at = $\{at(a_2) = \{integer\}\}$.
an = $\{an(a_1) = scr, an(a_2) = width\}$.
av = $\{av(a_1) = "\ldots", av(a_2) = "\ldots"\}$.
R = $\emptyset$.

For each document, which is used in the database system, a formal document model is created. Abstract directories deal with sets of documents. They provide a mechanism to create directories, hierarchies or even ontology for XML documents. I call them directories because fundamentally they are similar to directories of a file system. However, they provide a logical way for organizing documents, so there is no connection between abstract directories and directories of the file system. Abstract directories are based on the formal directory model, which can be defined as follows.

The ***formal directory model*** can be defined by the following relational structure.

$<DM,D,AV,dt,dn>$.

- DM is a finite set of formal document models of a database.
- $D \subseteq 2^{DM}$ is a finite set of abstract directories of the database.
- AV is finite set of atomic values.
- $dt:D \rightarrow AV$ is a partial map associating each directory with a type label.
- $dn:D \rightarrow AV$ is a partial map associating each directory with a name label.

Naming conventions:

- If $d_1, d_2 \in D$ and $d_1 \subseteq d_2$, than d_1 is a subdirectory of d_2 .
- If there is no d_3 for which $d_1 \subseteq d_3 \subseteq d_2$, than d_1 is a direct subdirectory of d_2.
- If $d_1, d_2 \in D$ are directories than formal document models of the two directories are $d_1 \cup d_2$.

Axioms of the structure are the followings:

1. If d_1 and d_2 are direct subdirectories of d_3, than the name label of d_1 and d_2 cannot be equal.
2. For each $dm \in DM$ document models $\{dm\} \in D$. In other words, all document models can be regarded as a directory consisting of only itself.

The conceptual level is a special kind of directory service. If directories and dependencies between directories are explicitly given, than the definition describes the traditional directory service of a file system. Maybe a little bit extended one, because directories can be typed and type constrained. However, if directories (set of XML files) are given implicitly, for example by logical statements or queries, than the definition is quite similar to a hierarchical view system of a relational database management system.

Constraints are special statements which force the structure of both the documents and the directories of a database. In this sense, abstract directories provide more well structured directories than directories of a simple file system. Since constraints are based on the constraint language, they will be discussed in details at the language level.

Based on these elements, an XML database and the XML database management system can be easily defined:

- The ***XML Database*** is a $<CM,C>$ pair where CM is a directory model and C is a set of constraints.
- The ***XML Database Management System*** is a $<<CM_1, C_1>,<CM_2, C_2>\ldots <CM_N, C_N>$» set of databases.

6. Language Level

Language level is based on two fundamental languages: document level language, directory level language. Both languages are lower-level logics, with Kripke or description logical semantics (or even with relational algebraic semantics). Document level language is SDL (Structured Document Logic), which is discussed in details in [18, 19]. Syntax and semantics of both languages are based on roles and concepts. Concepts, denoted by '*C*', represent subsets of nodes of the document model or subset of document models of the directory model, whilst roles, denoted by '*R*', are binary relations between nodes or document models. Syntax of both the document and the directory languages can easily be described by the following rules. For better readability basic elements of the syntax are written by bold characters.

Document Language (SDL):

R_{SDL} ::=***child*** | ***next*** | ***attribute-of*** | ***self*** R_{SDL} | ***inverse*** R_{SDL} | ***infinite*** R_{SDL}

C_{SDL} ::=***root*** | ***name*** '*name-symbol*'| ***value*** '*value-symbol*'| ***type***'*type-symbol*'| ***all-nodes*** | ***all-attributes*** | C_{SDL} ***and*** C_{SDL} | ***not*** C_{SDL} | ***all*** $R_{SDL}.C_{SDL}$ | $=N\,R_{SDL}.C_{SDL}$ | $<N\,R_{SDL}.C_{SDL}$

There are four kinds of simple roles, '*child*' and '*next*', describing children relation and next partial map of the document model. There are also two role constructors, '*inverse*' and '*infinite*' which represent inverse and transitive closure relations. There are three basic concept constructors. *{a}* does not represent a syntactic form, but the abbreviation of one piece of atomic concept which must be equal with an atomic predicate of the document model. Similarly, the term '*type*' or

‘*attribute-name*’ must be followed by a concrete type or attribute name, which must be found in the document model. The third simple role,’*attribute-of* ’, associates the structure of the document with the attributes, whilst ‘*value-of* ’ represents the connection between attributes and values. ‘*every*’ and ‘*none*’ are the universal and bottom concepts of the logic, ‘*and*’ ‘*or*’ ‘*not*’ are the basic logical operators whilst ‘*all*’ and ‘*some*’ denote universal and existential quantification over a role. ‘$= N$’ is a special quantification meaning that there are exactly ‘N’ pieces of elements which are in ‘C’. Comparison between document language and Xpath can be found among the others in [19].

Directory level language argues on formal directory models. Similarly, to the document language, it is based on concepts, denoted by ‘C_{DIR}’, and roles, denoted by ‘R_{DIR}’. Syntax of the directory language is based on the following rules.

Directory language:

R_{DIR} ::=***subdir*** | ***inverse*** R_{DIR}.

C_{DIR} ::= C_{SDL} | ***subdir-name*** *{value}*| ***subdir-type*** *{type}*| ***all*** | ***none*** |

C_{DIR} ***and*** C_{DIR} | C_{DIR} ***or*** C_{DIR} | ***not*** C_{DIR} | ***all*** $R_{DIR}.C_{DIR}$ | ***some*** $R_{DIR}.C_{DIR}$.

Directory level language is quite similar to the document level one, but it argues over set of documents instead of set of tags. ’*all*’, ’*and*’, ’*or*’, ’*not*’, ’*some*’, ’*none*’, ’*inverse*’ are basically the same as in the document language. ‘*subdir*’ represents the direct subdirectory relationship, ‘*subdir-name*’ and ‘*subdir-type*’ choose directories with the specified type or name label.

Document and directory languages represent the basic building blocks for both the query and the constraint languages of the XML database management system. They can be described by the following rules.

Query language:

QL ::= ***select tags*** C_{SDL} ***from*** C_{DIR} | ***select documents*** C_{DIR} ***from*** *{database-name}*

Constraint language:

CL ::= C_{DIR} ***implies*** C_{DIR} | C_{DIR} ***if and only if*** C_{DIR}

Query language contains two kinds of expressions. Traditional queries can be realized by the ‘*select tags…*’ statements. It can first find a set of documents from the database, but it is more common to find one specific document. Secondly, the query selects some tags of the selected documents. Query language can also be used to realize a classical search (‘*select documents…*’ statement). In this case, the statement finds some specific documents from the database and the result of the search is the set of documents not a set of tags. Similarly to query language, constraint language also has two kinds of expressions. ‘i*mplies*’ statement realizes a subsumption style constraint expression, whilst ‘*if and only if*’ represents an equivalence.

In order to define a formal semantics of the document language, an I interpretation function is considered, which assigns to every concept expression a set of nodes of a given ‘d’ document model, and to every role a binary relation over $\{N \cup AN\} \times \{N \cup AN\}$. Similarly, formal semantics of the directory language is given with the help of a J interpretation function, which assigns to every concept expression a set of directories of a directory model, and to every role a binary relation over

$D \times D$. In the followings $C^I_{SDL}[d]$ denotes the interpretation of a C_{SDL} expression at '*d*' document modes and '#' denotes cardinality of a set.

Semantics of the document language (SDL):

child$^I = c$

next^I = {<x,y>, x,y∈N | y = n(x)}

attribute-of^I = {<x,y>, x∈N, y∈AN | y∈a(x)}

(***self*** $R)^I$ = {<x,y>∈(N∪AN)×(N∪AN)| <x,y>∈R^I or x=y}

(***inverse*** $R)^I$ = {<x,y>∈(N∪AN)×(N∪AN) | <y,x>∈R^I}

(***infinite*** $R)^I = \cup_{j>=1}(R^I)^j$, transitive closure of R^I

root^I ={r}.

name 'name-symbol'^{I} ={n∈N∪AN | 'name-symbol'=an(n) or 'name-symbol'=nn(n)}.

type 'type-symbol'^{I} ={n∈N∪AN | 'type-symbol'∈at(n) or 'type-symbol'∈nt(n)}.

value 'value-symbol'^{I} ={n∈AN | 'value_name'=av(n)}.

all-nodes$^I = N$.

all-attributes^I =AN.

(C_1 ***and*** $C_2)^I = C^I_1 \cap C^I_2$.

(***not*** $C)^I$ =(N∪AN) \ C^I.

(***all*** $R.C)^I$ ={v∈{V∪AN} | ∀w. <v,w>∈R^I implies w∈C^I}.

(= N $R.C)^I$ = {v∈{V∪AN} | #{∃w. <v,w>∈R^I and w∈C^I}=N}.

(<N $R.C)^I$ = {v∈{V∪AN} | #{∃w. <v,w>∈R^I and w∈C^I}<N}.

Semantics of the directory language:

C^J_{SDL} = {{d} | d∈DM and $C^I_{SDL}[d] \neq \emptyset$}, where C_{SDL} is a concept expression of SDL.

subdir-name name^J = {d∈D | name=dn(d)}

subdir-type type_name^J = {d∈D | type_name=dt(d)}

subdir^J = {d_1, d_2∈D | d_1 is a direct subdirectory of d_2}

(***inverse*** $R)^J$ = {<w,v>∈D×D | <v,w>∈ R^J}

all$^J = D$

none$^J = \emptyset$

(***not*** $C)^J = D \setminus C^J$

(C_1 ***and*** $C_2)^J = C^J_1 \cap C^J_2$

(C_1 ***or*** $C_2)^J = C^J_1 \cup C^J_2$

(***all*** $R.C)^J$ = {d∈D | ∀w. <d,w>∈R^J implies w∈C^J}

(***some*** $R.C)^J$ = {d∈D | ∃w. <d,w>∈R^J and w∈C^J}

Semantics of the query language:

(***select tags*** C_{SDL} ***from*** $C_{DIR})^J = \cup_i(C^I_{SDL}[dm_i])$, for each $dm_i \in C^J_{DIR}$ formal document model.

(***select documents*** C_{DIR} ***from*** {database-name})$^J = \cup_i d_i$ for each $d_i \in C^J_{DIR}$ directory at the database-name database context.

Semantics of the constraint language:

$(C^1_{DIR}$ ***implies*** $C^2_{DIR})^J$ = *true if and only if,* $(C^1_{DIR})^J \subseteq (C^2_{DIR})^J$.

$(C^1_{DIR}$ ***if and only if*** $C^2_{DIR})^J$ = *true if and only if,* $(C^1_{DIR})^J = (C^2_{DIR})^J$.

As a simple example, imagine that confidential XML documents of a company are marked by a special confidentiality notes (*Fig. 2*). Assume that there are three kinds of documents: document with no confidentiality note, highly and lowly confidential documents. Confidentiality of a document is expressed by an XML tag, high and low is simply an attribute of the tag. One can also assume that the documents are stored in abstract directories. '*high_conf_dir*' stores documents which are highly confidential, whilst lowly confidential documents are stored in the '*low_conf_dir*' directory (see *Fig. 3*). The non-confidential documents can be stored anywhere in the database.

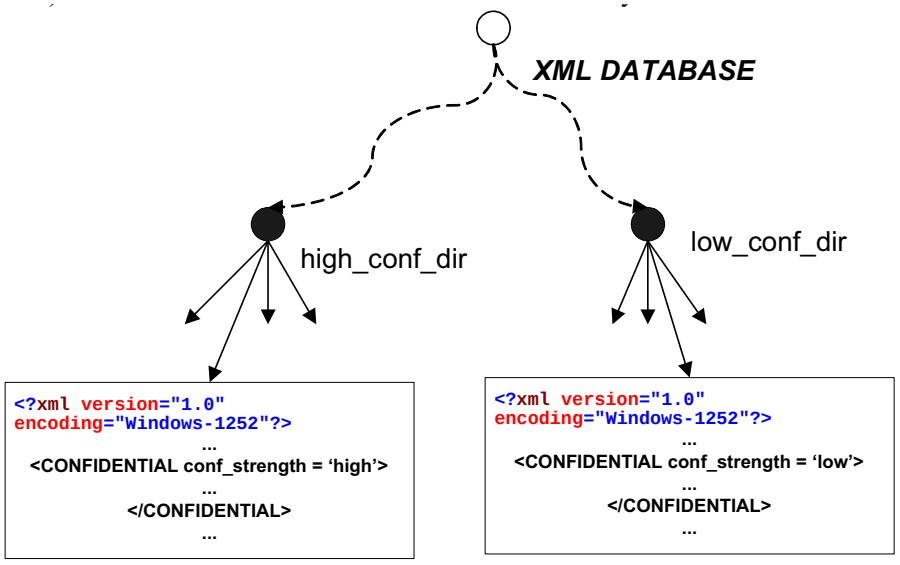

Fig. 3. Example for abstract directories

The following query and constraint statements demonstrate application examples for the query and constraint languages.

- '***select documents name*** *confidential* ***from*** *actual_database'* statement queries those documents which contain a tag called confidential.
- '***select documents root and all***(***infinite child***).(***not*** (***name*** *confidential*)) ***from*** *actual_database'* statement queries those documents which do not contain a tag called confidential.
- '***select documents name*** *confidential* ***and some*** (***attribute-of***).(***name*** *conf_strength* ***and value*** *high)* ***from*** *actual_database'* statement queries those documents which contain a tag called confidential and this tag has a

conf_strength attribute with high value. As it was mentioned previously, these are the highly confidential documents.

- *'**select documents name** confidential **and some** (**attribute-of**).(**name** conf_strength **and value** low) **from** actual_database'* statement queries those documents which are lowly confidential.
- *'**select documents subdir-name** high_conf_dir **from** actual_database'* statement queries all documents of the high_conf_dir directory.
- *'**select documents subdir-name** high_conf_dir **and all subdir.**(**name** confidential **and some attribute-of.**(**name** conf_strength **and value** high)) **from** actual_database'* statement queries all documents from the high_conf_dir directory which are really highly confidential.
- *'**select documents subdir-name** high_conf_dir **and some subdir.**(**name** confidential **and some attribute-of.**(**name** conf_strength **and value** low)) **from** actual_database'* statement queries all documents from the high_conf_dir directory which are lowly confidential.
- *'**select documents subdir-name** high_conf_dir **and some subdir.**(**root and all**(**infinite child**).(**not** (**name** confidential)) **from** actual_database'* statement queries all documents from the high_conf_dir directory which are not confidential at all.
- *'**subdir-name** high_conf_dir **implies all subdir.**(**name** confidential **and some attribute-of.**(**name** conf_strength **and value** high))'* statement is a constraint which forces that all documents in the high_conf_dir directory must be highly confidential.
  - *'**subdir-name** low_conf_dir **implies all subdir.**(**name** confidential **and some attribute-of.**(**name** conf_strength **and**(**value** high **or value** low)))'* statement is a constraint which forces that all documents in the low_conf_dir directory must be lowly or highly confidential.

Certainly, this dozen of examples demonstrated only a very limited part of the expressive power of the constraint and query languages. A lot of other structures, like types, type hierarchies or inheritance could also be easily applied for abstract directories.

The language level also contains some elements of the other industrial application, like Xpath, DTD or Xschema. However, these elements can be little used in this database, because neither industrial application is able to use the abstract directory system. Therefore, they are only supported by a heuristic mechanism which translates statements of the industrial applications to expression fragments of the query or constraint languages. These fragments usually need to be extended to form complete and correct query or constraint statements.

Unfortunately a query and a constraint language of a database system are not really enough to write real industrial applications. For example, Oracle databases have full scale DDL (Data Definition Language) and DML (Data Modification Language), which contain imperative elements (like 'for', 'while-do' 'if-then' statements) even in the most basic programming environment (PL/SQL). Developing all possible services for real industrial applications would require a lot of efforts.

Hence, some of these services, especially the imperative elements, cannot be really handled by modal logic. For these reasons, language level of the architecture has been strongly integrated into an industrial imperative programming language (C#). Consequently, the necessary imperative elements are provided directly by C#, whilst additional services, like creating or deleting a subdirectory or loading an XML file, are supported by method calls of a simple API. *Table 1* demonstrates the most important method calls.

Table 1. Method calls of the API.

Method calls of the language level	
'execute-query'	'add-constraint'
'load'	'delete-constraint'
'delete'	'create-attribute'
'write-out'	'delete-attribute'
'create-document'	'create-directory'
'create-tag'	'delete-directory'
'delete-tag'	'create-database'
	'delete-database'

7. Database Engine

Database engine provides the basic algorithmic elements for evaluating a query expression or forcing a constraint statement. Since document and directory languages are modal logics over document and directory models, the working mechanism of the engine is based on the different variations of model checking.

- Basic model checking: We have a '*d*' document or directory model and a *C* concept expression. The question is whether $C^I[d]=\emptyset$ or $C^I[d]\neq\emptyset$. In other words, *C* is true or false for '*d*'. This service can be used when the document level statements are embedded into directory expressions.
- Querying in model checking: We have a '*d*' document or directory model and a *C* concept expression. The result of querying is nodes of the document model for which *C* is true. Querying is also interpreted for directory statements. If so, the result is those directories for which the expression is true. Querying provides the basic reasoning service for evaluating select statements.
- Subsumption or implication in model checking: An 'exp_1' expression implying 'exp_2' for a '*d*' model means that, if 'exp_1' is true for '*d*', 'exp_2' is also true for '*d*'. Consequently, implication is not examined for all possible models, only for several predefined ones. Implication can be interpreted

for both document and directory expressions and models. Implication is the basic reasoning service for writing 'implies' constraints.

- Equivalence in model checking: An 'exp_1' equivalent with 'exp_2' for a 'd' model means that, 'exp_1' is true for 'd' if and only if 'exp_2' is also true for 'd'. Equivalence can be interpreted for both document and directory expressions and models. Equivalence is the basic reasoning service for writing 'if and only if' constraints.

Of course the most important question is efficiency, which highly influences the industrial applicability of every theory. Fortunately, model checking can be solved in polynomial time for some of the modal logics. The situation is even better in our case, model checking and variations of model checking can be solved by strongly polynomial time algorithms [20]. This complexity is sufficient for model checking a document model; however model checking a whole directory model containing plenty of documents could be slow for industrial applications. Therefore different heuristics for evaluating expressions over directory models are being developed.

With the help of algorithms for both SDL and the directory level language, evaluating a query or constraint expression can be easily defined.

- '***select tags***C_{SDL} ***from***C_{DIR}': *Firstly,* 'C_{DIR}' must be evaluated in an XML database, which results a set of documents. Then 'C_{SDL}' must be evaluated over the set of documents, which results a set of tags. Both evaluations are realized by the querying in model checking.
- '***select documents***C_{DIR} ***from*** *{database-name}':* C_{DIR}' must be evaluated in the 'database-name' database, which results a set of documents. The evaluation is realized by the querying in model checking.
- 'C^1_{DIR} ***implies***$C^2{}_{DIR}$': It can be realized in two ways. On the one hand, 'C^1_{DIR}' and 'C^2_{DIR}' can be evaluated independently and the containment of the two set can be checked with the help of subsumption in model checking basic reasoning service. On the other hand, the whole expression can be evaluated at once, with the help of a simple model checking mechanism.
- 'C^1_{DIR} ***if and only if*** $C^{2'}{}_{DIR}$: It can also be realized in two ways. On the one hand, 'C^1_{DIR}' and 'C^2_{DIR}' can be evaluated independently and the equivalence of the two sets can be checked with the help of equivalence in model checking basic reasoning service. On the other hand, the whole expression can be evaluated at once, with the help of a simple model checking mechanism.

8. Implementation Issues

This section briefly introduces some of the implementation issues of the XML database. *Fig. 4* demonstrates UML class diagram of the database. For the sake of simplicity, several not very important classes are not represented explicitly; instead they are represented as a set of support classes.

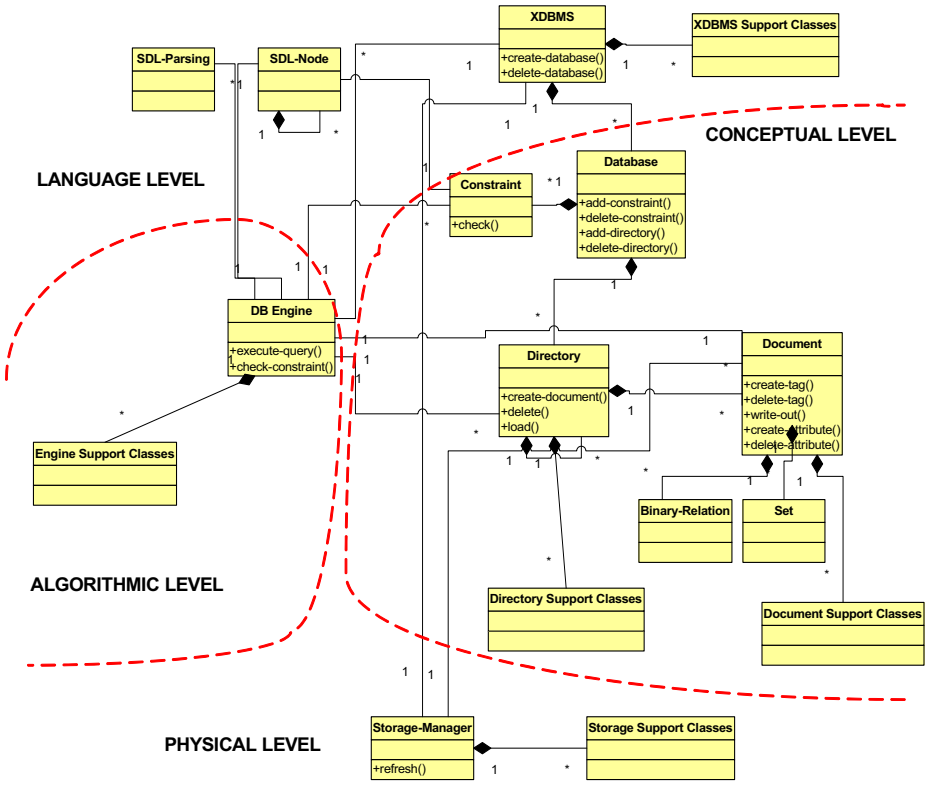

Fig. 4. UML diagram for the XML database.

At physical level, the most important class is the ‘Storage-Manager’. It controls the low level file activities with the help of some additional support classes (like classes for parsing XML documents…). Conceptual level contains classes for each important element. ‘Document’ class stores the formal document model with the help of some additional support classes, like ‘Binary-relation’ or ‘set’. Documents are stored directly in the tables which were introduced at the algorithmic aspects of SDL. Directories are represented by ‘Directory’ classes. The hierarchy between directories is stored as an object graph, which is extended with some elements so that it could be directly used in the database engine. ‘Database’ class represents databases of the architecture. Each database contains a set of directories and a set of constraints as well, which are implemented by the ‘Constraint’ class. The main class of the algorithmic level is the ‘DB-Engine’ which implements the basic reasoning services for both SDL and the directory language. It also implements the evaluation mechanism for both the query and the constraint languages. Representation of query and constraint expressions is realized by the ‘SDL-Node’ class at the language level. Parsing of the expressions is supported by the ‘SDL-Parsing’

class. The main interface of the whole architecture is the 'XDBMS' (XML Database Management System) class. It controls directly or indirectly the life-cycle and persistence of each element. C# programmers are able to get the first reference to the architecture with the help of 'XDBMS' class.

9. Conclusion

This paper has analysed the possibilities of realizing an XML database system by a pure modal logical formalism. The formalism is able to fully capture both the document and directory representations, both the directory and document level languages, both the query and constraint languages. Unfortunately, the logical approach is not able to describe totally a DDL or DML language. However, both languages can be strongly supported by the logical approach if they are strongly integrated with a host language environment which implements imperative elements. Comparing to other industrial XML databases, the logical approach has several benefits. It provides well-defined syntax, semantics and efficient computational elements for both the query and for the constraint language. Other approaches like Xquery provide only well-defined syntax. Hence, the logical approach makes the database and the database engine very simple from both technical and theoretical points of view, because only variations of the model checking algorithm should be developed. The previous section has demonstrated that the implementation of the database is very simple (a few dozens of classes) which is much simpler than the usual few hundreds of classes of an industrial database. Certainly, there are some elements of an industrial database, like transaction handling or multi-user support, which cannot be supported by the logical approach. However, the same situation is true for relational or deductive databases as well. Cores of these databases are developed by powerful theoretical approaches which are either relational algebra or Prolog, whilst additional mechanisms like transaction handling or multi-user support are developed by different ideas. In this sense, it does not seem to be a huge drawback if my logical approach cannot support for example transaction handling. In conclusion, I would dare to make the statement that our logical approach provides a simple and powerful environment for developing the core architecture of an XML database.

References

[1] GAROFALAKIS, M. N.– GIONIS, A.– RASTOGI, R.– SESHADRI, S.– SHIM, K.. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. *In Proceedings of ACM SIGMOD Conference on Management of Data*, (2000), pp. 165–176.

[2] FERNANDEZ, M. F.–MORISHIMA, A.– SUCIU, D.. Efficient Evaluation of XML Middle-ware Queries. *In SIGMOD '01*, (2001)

[3] XML Path Language (XPath), Version 1.0, *W3C Recommendation*, http://www.w3.org/TR/xpath, (1999)

[4] XQuery 1.0: An XML Query Language, *W3C Working Draft*, http://www.w3.org/TR/xquery/{#}nt-bnf , (2002).
[5] BAADER, F. – NUTT, W., Basic Description Logics. In the Description Logic Handbook, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, *Cambridge University Press*, pp. 47-100, (2002).
[6] NARDI, D. – BRACHMAN, R. J., An Introduction to Description Logics. In the Description Logic Handbook, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, *Cambridge University Press*, pp. 5-44, (2002).
[7] BONIFATI, A. – CERI, S., Comparative Analysis of Five XML Query Languages, *ACM SIGMOD Record* vol. 29 , Issue 1, pp. 68 – 79 (2000)
[8] GOLDMAN, R. – MCHUGH, J. Widom, From Semistructured Data to XML: Migrating the Lore Data Model and Query Language, *In Proc. Of the 2nd International Workshop on the Web and Databases*, (1999)
[9] CERI, S.– COMAI, S.– DAMIANI, E.– FRATERNALI, P.– PARABOSCHI, S.– TANCA, L., XML-GL: A Graphical Language for Querying and Reconstructing WWW Data, *In Proc. of 8th Int. World Wide Web Conference, WWW8*, (1999).
[10] YOSHIKAWA, M.– AMAGASA, T.XREL: a Path-based Approach to Storage and Retrieval of XML Documents Using Relational Databases, *ACM Transactions on Internet Technology (TOIT)* vol. 1, issue 1 pp. 110 – 141, (2001)
[11] DEUTCH, A.– FERNANDEZ,M.– FLORESCU, D.– LEVY, A.– SUCIU, D., A Query Language for XML, *In Proc. of 8th Int. World Wide Web Conference, WWW8*, (1999).
[12] FRASINCAR, F. – HOUBEN, G-J.– PAU"C. XAL: An Algebra For XML Query Optimization, *Thirteenth Australasian Database Conference (ADC2002)* (2002)
[13] FRASINCAR, F. – HOUBEN, G-J.– PAU"C." XAL: An Algebra For XML Query Optimization, *Thirteenth Australasian Database Conference (ADC2002)* (2002)
[14] JAGADISH,V.– LAKSHMANAN, L. V. S.– SRIVASTAVA,D.– THOMPSON, K., *TAX: A Tree Algebra for XML*, In: *Proceedings of 8th International Workshop on Databases and Programming Languages*, Rome, Italy, September (2001)
[15] MING, Z.– JINGTAO, Y., XML Algebras for Data Mining, Data Mining and Knowledge Discovery: Theory, Tools, and Technology VI. Edited by Dasarathy, Belur *V. Proceedings of the SPIE*, vol. 5433, pp. 209-217 (1999)
[16] ABITEBOUL, S., Querying Semistructured Data, *In Proceedings of the International Conference on Database Theory*, pp.1-18, (1997)
[17] CALVANESE, D.–GIACOMO, G. – LENZERINI, M., Modeling and Querying Semi-structured Data, *Network and Information Systems*, 2(2):253-273, (1999)
[18] SZEGŐ, D.: Structured Document Logic, *Periodica Polytechnica* Vol. 47. No. 3-4, pp. 311-324 (2003).
[19] SZEGŐ, D.: Using Description Logics in Web Document Processing, *SOFSEM (SOFtware SEMinar)* vol. II. pp. 256-263, (2004)
[20] SZEGŐ, D.: A Fast and Efficient Method for Processing Web Documents, *ICCS (International Conference on Computational Science)* LNCS 3038 pp. 553–556. (2004)