Using taDOM Locking Protocol in a Functional XML Update Language

Pavel Strnad and Pavel Loupal

Department of Computer Science and Engineering
Faculty of Electrical Engineering, Czech Technical University
Karlovo náměstí 13, 121 35 Praha 2, Czech Republic
`strnap1@fel.cvut.cz, loupalp@fel.cvut.cz`

Abstract. In this paper we deal with a particular type of database systems – native XML database systems. For this category of systems we discuss potential application of the taDOM locking protocol implemented in a functional update language – XML-λ. By combination of these theoretical approaches we obtain a solution for querying and updating XML data that can be implemented in a native XML database system with transaction support. We present an LL(1) translation grammar for transformation of queries written in a functional language into sequence of Document Object Model API calls.

1 Motivation

Currently, our research group works on development of a native XML database system that uses XQuery and should also use the XML-λ query language. Therefore we are interested in properties and interconnections between these two artifacts. XQuery represents de-facto industrial standard in querying and XML-λ is a proposal of our group based on simply typed λ-calculus.

In this paper we submit a proposal for transformation of XML-λ statements into a list of DOM operations with ensured transaction isolation through the DOM-based locking protocol – taDOM. We plan to use this solution for extending our native XML database system in the future.

2 Introduction

The crucial property of modern database management systems (DBMS) [11] is concurrent user access. In this work we discuss it in context of a specific type of database systems – native XML database systems. Such systems are primarily used for storing XML data in their original form instead of mapping its structures into e.g. objects or relations.

We outline an existing locking protocol that was developed for XML data – taDOM [14] – and use it for a particular functional query and update language – XML-λ [19, 20]. It is a proposal of a functional framework for querying and manipulating XML data. It is established on type system theory and utilizes simply typed λ-calculus as a base for specification of a query language. This project is currently in phase of development and in future we plan to include an XML-λ module into native XML database systems we currently work on – CellStore [25] and ExDB [1].

The approach we present in this paper is as follows: we transform basic update operations into standard DOM [24] operations that are supported by taDOM and so we can introduce transactional behavior into the language. Formally, we use an LL(1) translation grammar for converting XML-λ expressions to DOM API method calls.

The main contribution of this paper lies in combining the taDOM locking protocol and XML-λ query/update language. We offer a proposal how to interface referenced locking protocol and low-level update operations in given language using DOM operations. This work continues in the topic that we have opened in [17]. In this text we clarify more the update facility of the framework and design a translation grammar that provides the solution for transformation of XML-λ update statements into DOM operations.

The rest of the paper is structured as follows: Section 3 gives a short overview about concurrency and related issues in database systems, Section 4 then outlines the idea of the taDOM locking protocols family. Basic concept of XML-λ with its key update features is briefly introduced in Section 5. The main part of this work that describes our proposal for mapping of DOM operations to the query language is presented in Section 6. In Section 7 we gather a list of certain related papers available. Finally, we conclude with ideas for our future research in Section 8.

3 Concurrency in Native XML DBMS

Common requirement for a database management system is the concurrency control. There are four well-known properties for a transactional system known as *ACID* [11]. Transaction is generally a unit of work in a database. ACID properties are independent on a database (logical) model (i.e. it must be kept in all transactional database systems).

Isolation of transactions in a database system is usually ensured by a locking protocol. Direct application of a locking protocol used in relational databases does not provide high concurrency [15, 22] (i.e. transactions are waiting longer than it is necessary).

We show a huge difference between locking protocols for RDBMS and native XML database system on a small example. Let us have two lock modes: Shared mode and Exclusive mode. The granularity of exclusive lock in RDBMS is typically the row (or record) [6]. In a native XML database we have much more possibilities whether to lock a node or a whole subtree. Hence, we have more choices what to lock and for how long time. These protocols working on XML data extend the basic locking protocol. The basic protocol provides only two types of lock modes, but taDOM3+ has twenty lock modes. More lock modes imply more complexity in a protocol algorithm. Also to prove whether the protocol is correct is a harder problem.

We suppose only well-formed transactions and serializable plan of update operations [6]. All protocols quoted in this paper satisfy these requirements. We call locking protocols for native XML databases simply *XML-locking protocols* in this paper. The most of these XML-locking protocols are based on the basic relational locking protocols. Hence, XML-locking protocols inherit most of the features, e.g. two-phase locking to ensure serializability.

To design a good locking protocol which minimizes the number of suspended transactions is a challenge because it is much more complex than in RDBMS. It requires new lock modes for individual elements and also for the axes in an XML document [22]. The locking mechanism depends on the query language used, concretely on the atomic operations of the query language and also on the context of these operations.

4 Locking Protocols

Actual research in the area of locking protocols is concentrated rather to DOM model and its methods of approaching individual parts of an XML document. Probably the most advanced research in this topic is carried out at the University of Kaiserslautern in Germany [16, 14, 15]. The researchers are working on XTC (XML Transaction Coordinator) Project [3] – a system which implements several different algorithms of transactional processing on XML data.

XTC project uses extended DOM model (it is called *taDOM*) as a basis for transactional processing.

4.1 taDOM Family of Locking Protocols

The first version of the protocol was denominated as taDOM2. Its improved version is then called taDOM2+. Both of these protocols work with DOM Level 2 operations (about 20 methods, see [23]). Next generation of taDOM locking protocols are taDOM3 and taDOM3+. As expected, these protocols correspond to DOM Level 3 model. The XTC project also provides detailed use cases for these protocols (36 use cases) which completely describe locking scenarios for each operation.

Each of taDOM locking protocols is specified by:

– Compatibility matrix
– Conversion matrix
– Use cases for DOM operations

The compatibility matrix is used when the transaction *t1* is requesting for lock *l1* on a node *n* and there is a lock *l2* of the transaction *t2*. The locking algorithm finds the row *l1* and column *l2* in the compatibility matrix and makes a decision whether to lock (+) or not (-). Table 1 describes Compatibility Matrix for edge locks (ER - edge read, EU - edge update and EX - edge exclusive).

	-	**ER**	**EU**	**EX**
ER	+	+	-	-
EU	+	+	-	-
EX	+	-	-	-

Table 1. The Compatibility Matrix of the Edge Locks

getNode(*nodeID*) **returns** *Node*					
Scenario 0-1 for taDOM3+ Lock Requests:					
Node	Lock	PSE	NSE	FCE	LCE
CN	NR	-	-	-	-

Table 2. Lock Scenario for DOM Operation *getNode*(*nodeID*)

The conversion matrix is used when the transaction *t1* is requesting for lock *l1* on a node n and there is a lock *l2* of the same transaction *t1*. Locking algorithm finds the row *l1* and column *l2* in the conversion matrix and converts the lock mode of the node. Hence, each transaction has at the most one lock on each node.

Use cases describe semantics of the locking protocol with regard to DOM operations. Table 2 contains description of the DOM operation *getNode*(*nodeID*). When *getNode*(*nodeID*) operation is called then the locking mechanism has to put the lock of type NodeRead (NR) on the context node(CN). PSE, NSE, FCE, LCE are abbreviations for previous sibling edge, next sibling edge, first child edge, last child edge. The *getNode*(*nodeID*) operation does not put locks on these virtual edges (-).

We consider only taDOM3+ in the next sections of this paper. This protocol is up-to-date nowadays, because it reflects today's needs and was formally checked[1]. The taDOM3+ locking protocol has also really low overhead (minimizes access to the storage) [14].

taDOM3+ protocol provides level 2.99 of isolation [4, 15]. It means that phantom reads[2] are not covered. Therefore it is necessary to do a small extension to these protocols by adding navigation edges to avoid existence of phantom reads. We need to define an additional mechanism – edge locks. To apply edge locks the authors had to extend the XML document model and added new edges between nodes – virtual edges. The compatibility matrix of these locks is discussed in more details in [15].

taDOM Model Structure The tree-like structure in taDOM is enriched by two new node types: *attributeRoot* and *string* [14]. This representational en-

[1] Valenta and Siirtola [21] made a formal proof of the protocol correctness. They verified the taDOM locking protocol using model-checking.

[2] Phantom read happens when new rows added by a transaction are visible from another transaction

hancement does not influence user operations and their semantics on the XML document, but is solely exploited by the lock manager to achieve certain kinds of optimization when an XML document is modified in a cooperative fashion [15].

– *attributeRoot* separates various attribute nodes from their element node. Instead of locking all attribute nodes separately they are locked all together by placing the lock to attributeRoot – concurrency of attribute processing is not allowed.
– A *string* node is attached to the respective text node and only contains the value of this node. It does not allow to block a transaction which only navigates across the node, although a concurrent transaction may have modified the text (content) and may still hold an exclusive lock on it.

Lock Modes The taDOM3+ protocol provides a set of lock modes for the nodes as well as for the edges. Edge locks are used to cover phantom reads in an XML document in order to allow desired level of concurrency. The lock modes together with their mutual relationships (expressed as compatibility matrices) provide concurrency and also preserve the expected ACID properties (especially the level of isolation).

5 Updating XML

There are various theoretical proposals, experimental implementations and defacto standards for XML update languages. As of the publication of the XML 1.0 standard [8] the efforts had been focused on querying such structured data. Approaches for updating have gained more attention in past few years.

Existing papers dealing with updating XML are mostly related to XQuery [7]. The need for introducing updates into XQuery is also considered as one of the most important topics in the further development of the language [10]. As a result, new specification of the XQuery Update Facility [9] was proposed.

In this paper we deal with our approach for querying and updating XML data – XML-λ [19, 20]. It is a framework for manipulating XML based on a type system and simply typed λ-calculus. It is a cornerstone of our long-term research that was invented not only for definition of an update language for XML but also for a broader exploitation, for example heterogeneous data integration or description of denotational and operational semantics of various languages. Indeed, in this paper we focus on using it as a query and update language for XML.

In following paragraphs we expect that reader is familiar with basic concept of the XML-λ Framework. Nevertheless, we repeat its basic concept for convenience.

5.1 Updates in General

Query execution has in general the following structure: (1) Declaration of variables, (2) Evaluation of variables and tree traversal and (3) Output construction.

For read-only systems and even parallel user access it is perfectly sufficient but for systems with update support it is necessary to use a different approach.

Usually, for particular update operations (in the world of native XML database systems) a structure called "pending update list" is utilized. This structure contains ordered list of fundamental modification operations to be carried out at the end of each transaction. Hence, the execution of an update operation (insert, delete, replace) then follows these two steps: (1) node locking and (2) appending an appropriate update operation to the list.

At the end of each transaction the pending update list is processed and all nodes locked by the transaction are unlocked at its end (both in case of commit or abort). This approach is applied both in XQuery and XML-λ[3]. In following sections we cover our solution based on XML-λ in detail.

5.2 XML-λ Framework Basics

XML-λ is a functional framework for manipulating XML. The original proposal [19, 20] defines its formal base and shows its usage primarily as a query language for XML but there is a consecutive work that introduces updates into the language available in [17].

In XML-λ there are three important components related to its type system: *element types*, *element objects* and *elements*. We can imagine these components as the data dictionary in relational database systems. Note also Figure 1 for relationships between basic terms of W3C standards and the XML-λ Framework.

Element types are derived from a particular DTD and in our scenario they cannot be changed – we do not allow any schema changes but only data modifications. For each element defined in the DTD there exists exactly one element type in the set of all available element types (called T_E).

Consequently, we denote E as a set of *abstract elements*. Set members are of element types.

Element objects are basically functions of type either $E \rightarrow String$ or $E \rightarrow (E \times \ldots \times E)$. Application of these functions to an *abstract element* allows access to element's content. *Elements* are, informally, values of *element objects*, i.e. of functions. For each $t \in T_E$ there exists a corresponding t-object.

For convenience, we add a "nullary function" (also known as 0-ary function) into our model. This function returns a set of all abstract elements of a given element type from an XML document.

Finally, we can say that in XML-λ the instance of an XML document is represented by a set E and set of respective t-objects.

Example. Let us consider an example DTD and a fragment of an XML instance shown in Figure 2. For given schema we derive element types as follows: $BIB : BOOK*$, $BOOK : (TITLE, AUTHOR+, PRICE)$, $AUTHOR : (LAST, FIRST)$, $LAST : String$, $FIRST : String$, $TITLE : String$, $PRICE :$

[3] Note that both XQuery and XML-λ are functional languages.

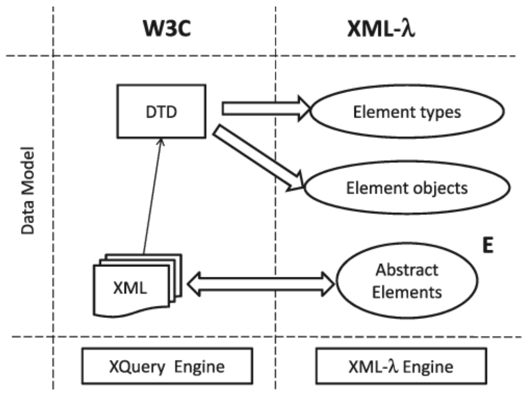

Fig. 1. The Relationship Between W3C and XML-λ Models

String.

Then, we define functional types – designated as t-objects: $BIB : E \rightarrow 2^E$, $BOOK : E \rightarrow (E \times 2^E \times E)$, $AUTHOR : E \rightarrow (E \times E)$, $TITLE : E \rightarrow String$, $LAST : E \rightarrow String$, $FIRST : E \rightarrow String$, $PRICE : E \rightarrow String$.

```
<!ELEMENT bib  (book* )>                  |  <bib>
<!ELEMENT book  (title,  author+,         |    <book>
          price )>                        |      <title>TCP/IP Illustrated</titl
<!ELEMENT author  (last, first )>         |      <author>
<!ELEMENT title  (#PCDATA )>              |        <last>Stevens</last>
<!ELEMENT last  (#PCDATA )>               |        <first>W.</first>
<!ELEMENT first  (#PCDATA )>              |      </author>
<!ELEMENT price  (#PCDATA )>              |      <price>65.95</price>
                                          |    </book>
                                          |    ...
```

Fig. 2. Example DTD and Fragment of a Valid XML Instance

Having look at the Figure 2 we can see that there are 7 abstract elements (members of $E' \subset E$). Now, for instance, the *price*-object is defined exactly for one abstract element (the one obtained from `<price>65.95</price>` element) and for this abstract element it returns value ”65.95”.

Let us consider a query that returns all books with price higher than 100. This query is written in XML-λ as:

```
xmldata("bib.xml")
lambda b ( /book(b) and b/price > 100))
```

Here, we do not depict the query evaluation process in detail but it is described sufficiently in [19, 20].

5.3 Fundamentals of Updates

By introspecting the basics of the framework outlined in Section 5.2 – especially the idea of element objects we can see that by updating an XML document we modify actual domains of element objects (i.e. partial functions defined on E) and their ranges.

6 Locking Protocol Mappings

This section describes our solution for translation of XML-λ statements into DOM API calls through a top-down parser directed by an attributed LL(1) translation grammar.

For easier specification of transformation between XML-λ primitives and DOM operations we define new operation $\diamond$:

$$f^{+}(v) = \{f^{1}(v), f^{2}(v), f^{3}(v), \ldots\}$$

$$f^{+}(v) \diamond g() = \bigcup_{u=1}^{\infty} \{g(f^{u}(v))\}$$

This operation is defined on sets. We can say that the $g()$ function is applied on each element of a set.

6.1 A Pinch of Translation Theory

We solve the problem of mapping by translation from one language to another. The straightforward approach is based on construction of an attributed translation grammar [5]. Then all queries written in XML-λ can be translated into a sequence of DOM operations.

Here we refer shortly to definition related to translation grammars – note that we use an attributed translation grammar, i.e. a context-free grammar augmented with attributes, output symbols and semantic rules.

The attributed translation grammar is 4-tuple $APG =< PG, A, V, F >$, where PG is a basic translation grammar $PG =< N, \Sigma, D, R, S >$. N is set of non-terminal symbols, Σ is set of terminals, D is set of output symbols, R is a set of grammar rules $A => \alpha$, where $A \in N$, $\alpha \in (N \cup \Sigma \cup D)*$ and S is the start symbol.

Remaining symbols are related to APG and have the following meaning

A is a finite set of attributes. It is divided into two disjoint sets for synthesized (denoted S) and inherited (denoted I) attributes.
V is a mapping that assigns a set of attributes to each non-terminal symbol $X \in N$
F is a finite set of semantic rules

The example stated in the following section is based on this formalism.

6.2 XML-λ to DOM Translation Grammar

We use the standard formal translation directed by an LL(1) parser where the formal translation is described by translation grammar as follows:

$$\begin{aligned}
N &= \{S, R_0, R_1, T\} \\
\Sigma &= \{/, sL, var\} \\
D &= \{\ \textcircled{s}, \textcircled{t}, \textcircled{c}\ \} \\
R &= \{\ S \rightarrow /\ R_0 | var R_1, \\
&\qquad R_0 \rightarrow sL\ \ \textcircled{s}\ \ T\ R_1, \\
&\qquad R_1 \rightarrow /\ \ \textcircled{c}\ \ sL\ T\ R_1 | \epsilon, \\
&\qquad T \rightarrow \textcircled{t}\ \}
\end{aligned}$$

Note that terminal symbols are output tokens from a lexical analyzer.
We proposed necessary attributes for translation $A = \{name, string\}$, where $I(T) = \{name\}$, $I(\textcircled{t}) = \{name\}$, $S(sL) = \{string\}$. Attributes are used for storing tag names in the process of translation.

Syntax and semantics of the translation grammar is described in Table 3.

Syntax	**Semantics**
$S \rightarrow /\ R_0 \vert var R_1$	
$R_0 \rightarrow sL\ \ \textcircled{s}\ \ T\ R_1$	T.name := sL.string
$R_1 \rightarrow /\ \ \textcircled{c}\ \ sL\ T\ R_1 \vert \epsilon$	T.name := sL.string
$T \rightarrow \textcircled{t}$	$\textcircled{t}$.name := T.name

Table 3. Syntax and Semantics Table

After translation the output symbols are rewritten in following way:

$$\begin{aligned}
&\textcircled{s} \rightarrow doc\ \diamond\ getDocumentElement()\ \diamond\ getChildNodes()^{+} \\
&\textcircled{t} \rightarrow \diamond\ getTagName(\textcircled{t}.\text{name}) \\
&\textcircled{c} \rightarrow \diamond\ getChildNodes()
\end{aligned}$$

Following example shows how we can transform XML-λ queries to DOM operations. These operations implicitly use taDOM3+ locking protocol synchronization primitives.

6.3 XML-λ Query Evaluation Example

Let us have a look at an example of a delete operation in the XML-λ language. Following statement deletes all books specified by given title:

```
xmldata("bib.xml")
delete( lambda b ( /book(b) and
          b/title = "TCP/IP Unleashed"))
```

We translate the inner expression of the statement

```
(/book(b) and b/title = "TCP/IP Unleashed")
```

The translation is based on a top-down method using expansion operation $\Rightarrow$. Expansion rule depends on the top terminal of the processed input string. Then we can use a standard LL(1) parser. Translation then starts as follows:

$$S \Rightarrow /\ R_0 \overset{R_0}{\Rightarrow} /\ sL\ \textcircled{s}\ \ T\ R_1 \overset{T}{\Rightarrow}\ /\ sL\ \textcircled{s}\ \textcircled{t}\ R_1 \overset{R_1}{\Rightarrow}\ /\ sL\ \textcircled{s}\ \textcircled{t}$$

By this derivation we have translated the first part of the expression – `/book(b)`. Then, we continue with the second part:

$$S \Rightarrow\ var\ R_1 \overset{R_1}{\Rightarrow} var/\ \textcircled{c}\ sLTR_1 \overset{T}{\Rightarrow} var/\ \textcircled{c}\ sL\ \textcircled{t}\ R_1 \overset{R_1}{\Rightarrow} var/\ \textcircled{c}\ sL\ \textcircled{t}$$

We get the translated string by omitting input symbols. We suppose that the semantic rules were applied during translation. In the input symbol `var` we saved the first part of the translation. The second part is concatenated with the first part through the variable `b`. The output of the translation is the following sequence of output symbols: $\textcircled{s}\ \textcircled{t}\ \textcircled{c}\ \textcircled{t}$.

We can rewrite these output symbols to taDOM operations and then we get:

$$doc \diamond getDocumentElement() \diamond getChildNodes()^{+} \diamond getTagName(\textcircled{t}.\mathrm{name}) \\ \diamond getChildNodes() \diamond getTagName(\textcircled{t}.\mathrm{name})$$

The main part of the update statement is the path expression. Now we have to select nodes which satisfy condition – *title* = *"TCP/IP Unleashed"*. The string comparison operation is not a DOM operation, so for purpose of this paper is omitted here.

The translation grammar described above can be directly used to ensure isolation of transactions in the XML-λ language.

7 Related Work

First, considering query and update languages, there are many papers and proposals. The most important specifications in the context of this work are the XML Query Language 1.0 Specification [7] and a Working Draft of the XQuery Update Facility [9]. They form de-facto standard in the world of XML.

Another branch of papers is focused on a specific database system and describes usually a complete solution seen from a wider perspective. It is quite a common practice that database groups at technical universities and similar institutes develop their own database systems. In the area of XML we can mention eXist [2] or Natix [12]. Finally, we should also mention CellStore [25] – a database system being developed at our department. Authors of these systems usually describe in detail the transformation of XQuery statements into their algebras.

Issues related to transactions and systems that support transactional behavior are covered as description of transaction protocols [6, 13, 14] or transactional benchmarking [18].

8 Conclusions and Future Work

We have shown an approach for introducing fundamental transactional operations into a functional query and update language for XML. Through the use of the taDOM protocol and the XML-λ Framework we have obtained a theoretical solution for native XML database systems that allows querying and updating XML data in a safe manner. We accomplish this by introducing an LL(1) attributed translation grammar for transformation of XML-λ statements into sequence of DOM API calls. This forms a solid base for our ongoing research – studying of mutual semantic transformations of queries written in one query language into another, e.g. conversion of XQuery queries into XML-λ and vice-versa.

Considering the fact that we have presented here only first sketch of the solution there is still a lot of clarification and experimental work ahead. In the near future we plan to design and implement a prototype of the XML-λ query engine into the CellStore database system. After that there are many open topics related to correctness and benchmarking of the prototype.

9 Acknowledgments

We would like to thank to Jiří Velebil for his valuable hints related to mathematical expressions we use in this paper to formulate formalisms in a (hopefully) clear way.

This work has been supported by the Ministry of Education, Youth, and Sports under Research Program MSM 6840770014.

References

1. ExDB Project Homepage. http://swing.felk.cvut.cz/~loupalp.
2. eXist Project Homepage. http://www.exist-db.org.
3. XTC Project. http://wwwdvs.informatik.uni-kl.de/agdbis/projects/xtc.
4. A. Adya, B. Liskov, and P. O'Neil. Generalized isolation level definitions. *ICDE*, 00:67, 2000.
5. A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation, and Compiling. II. Compiling*, volume II. Prentice-Hall, Upper Saddle River, NJ 07458, USA, 1973.
6. P. Bernstein and E. Newcomer. *Principles of Transaction Processing.* Morgan Kaufmann Publishers, 1st edition, 1997.

7. S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language, January 2007. http://www.w3.org/TR/xquery/.
8. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (third edition), February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/.
9. D. Chamberlin, D. Florescu, and J. Robie. XQuery update facility. http://www.w3.org/TR/2006/WD-xqupdate-20060711/.
10. D. D. Chamberlin. XQuery: Where do we go from here? In *XIME-P*, 2006.
11. C. J. Date. *An Introduction to Database Systems, 6th Edition.* Addison-Wesley, 1995.
12. T. Fiebig, S. Helmer, C.-C. Kanne, G. Moerkotte, J. Neumann, R. Schiele, and T. Westmann. Anatomy of a native xml base management system. *VLDB Journal*, 11(4):292–314, 2002.
13. J. Gray and A. Reuter. *Transaction Processing : Concepts and Techniques.* Morgan Kaufmann Publishers, 1st edition, 1993.
14. M. P. Haustein and T. Härder. A synchronization concept for the DOM API. In H. Höpfner, G. Saake, and E. Schallehn, editors, *Grundlagen von Datenbanken*, pages 80–84. Fakultät für Informatik, Universität Magdeburg, 2003.
15. M. P. Haustein and T. Härder. An efficient infrastructure for native transactional XML processing. *Data Knowl. Eng.*, 61(3):500–523, 2007.
16. M. P. Haustein, T. Härder, C. Mathis, and M. W. 0002. Deweyids - the key to fine-grained management of xml documents. In C. A. Heuser, editor, *SBBD*, pages 85–99. UFU, 2005.
17. P. Loupal. Updating typed XML documents using a functional data model. In J. Pokorný, V. Snášel, and K. Richta, editors, *DATESO*, volume 235 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
18. M. Nicola, I. Kogan, and B. Schiefer. An xml transaction processing benchmark. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 937–948, New York, NY, USA, 2007. ACM Press.
19. J. Pokorný. XML functionally. In B. C. Desai, Y. Kioki, and M. Toyama, editors, *Proceedings of IDEAS2000*, pages 266–274. IEEE Comp. Society, 2000.
20. J. Pokorný. XML-λ: an extendible framework for manipulating XML data. In *Proceedings of BIS 2002*, pages 160–168, Poznan, 2002.
21. A. Siirtola and M. Valenta. Verifying parameterized taDOM+ lock managers. *SOFSEM 2008*, pages 460–472, 2008.
22. P. Strnad and M. Valenta. Object-oriented Implementation of Transaction Manager in CellStore Project. *Objekty 2006, Praha*, pages 273–283, 2006.
23. The W3C Consortium. W3C homepage. http://www.w3.org.
24. The W3C Consortium. Document Object Model (DOM), 2005. http://www.w3.org/DOM/.
25. J. Vraný. Cellstore - the vision of pure object database. In V. Snásel, K. Richta, and J. Pokorný, editors, *DATESO*, volume 176 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.