XML-λ Type System and Data Model Revealed

Pavel Loupal

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

Abstract. Within this paper we provide formal description of a functional type system for modeling XML formatted data along with an annotated example of an XML document modeled using such approach. We discuss its advantages and drawbacks in comparison with existing solutions. This submission is a part of our long-term endeavor to propose, examine, and implement an environment for XML data management, especially utilized for XPath/XQuery semantics description.

1 Introduction

The Extensible Markup Language (XML) 1.0 [1] specifies both physical and logical structure of XML documents. For purpose of related specifications (e.g., XPath or DOM) and other XML applications is such low-level description too punctual and hence a number of more abstract *data models* has been proposed. The best known data models nowadays are the XML Information Set (XML Infoset) [4] and the XQuery 1.0 and XPath 2.0 Data Model [7].

This paper describes an alternative data model for XML that forms a part of the XML-λ Framework – a functional framework for modeling, querying and updating XML. The main goal of this work is to provide the reader very intimate knowledge of the Framework and cover all its key concepts and properties. We suppose that the accompanying example presents these facts in reasonable complexity.

Contributions. The main outcome of this paper for the reader should be deep insight into the type system concept and familiarization with the XML-λ data model. We have aimed our endeavor to provide following contributions:

- Formal description of the functional type system for XML.
- Annotated example of an XML document modeled using the XML-λ.
- Informal discussion of Framework's properties and brief comparison with existing data models.
- Outline of the proof-of-the-concept prototype implementation.

K. Richta, J. Pokorný, V. Snášel (Eds.): Dateso 2009, pp. 130–141, ISBN 978-80-01-04323-3.

Structure. The paper is organized as follows: Section 2 contains formal description of the XML-λ Framework and shows a simple XML document modeled using this approach. General properties, features, and drawbacks of the Framework along with a brief comparison with other data models are discussed in Section 3. Then, in Section 4, we outline our prototype implementation and finally conclude our contribution in Section 5 together with outlook for future work.

Related Work. The main area of interest that is discussed within this paper is the domain of data models for XML. There are two principal data models for XML in use nowadays; the first is the XML Infoset [4] and the second is the XQuery 1.0 and XPath 2.0 Data Model [7]. Both specifications are proposed by the W3C [15] and are employed in interrelated XML standards. Roots of the XML-λ Framework might be found in Pokorný's work [12, 13].

2 The XML-λ Framework

Under the term "framework" we understand both the data model and the query language based on this model. Here, we define the fundamental cornerstone of the framework, the functional type system for XML that is suitable for modeling XML — $\mathcal{T}_E$. Then, in Section 2.3, we show a detailed example of an XML document instance realized using the XML-λ Framework.

2.1 Overview

Our research is significantly influenced by the idea of a functional approach for XML published by Pokorný in [12] and [13]. We found his work very appealing and useful in wide range of applications related to XML. Our main motivation is the belief that it is possible use this framework for expressing semantics of various XML query languages in formally clear and understandable way by using a simply-typed lambda calculus and a general type system. Thus, we set out a goal to express the semantics of XPath and XQuery languages using this approach called XML-λ. We do not aim to propose a new query language for XML but we rather offer an alternative solution for evaluation existing query languages.

Informal Introduction. Figure 1 illustrates the relationship between W3C and XML-λ models. We can observe that the document schema expressed by DTD is in the Framework modeled with two sets; the first one is a set of *element types* available in XML schema and the second one is a set of *element objects* that stores information about document structure. Every document instance then comprises of a set of abstract elements (each is of an element type) and a set of element object instances. Note that these two sets are changing in time, so as the document changes.

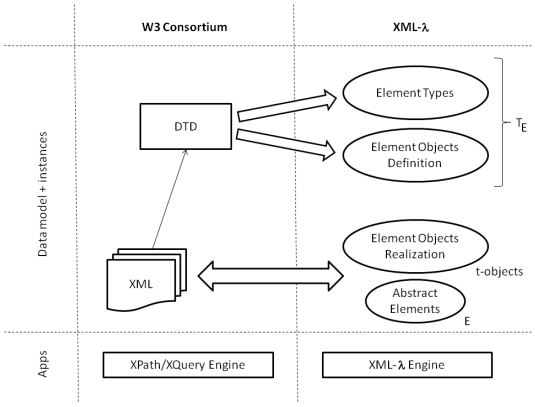

Fig. 1. The relationship between W3C and XML-λ models

2.2 Type System Definition

The main purpose of a *type system* is to prevent the occurrence of execution errors in a program[1] [2]. In this work we understand this definition as checking for correct nesting of XML document structure, using the type system for semantic query validation, and its further evaluation (including optimization).

But there is a unifying cornerstone for all type systems — each *type system* has a finite set of rules for type construction and set of respective types defined within this system. Following sections describe step-by-step process of definition of a functional type system for XML. We begin with a general functional type system and extend it with support for regular types. Finally, we propose the type system $\mathcal{T}_E$ that forms the core formalism in the XML-λ Framework.

Functional Type System First, we define a general functional type system $\mathcal{T}$. This definition basically follows the common approach presented for example in [16, p.143].

Definition 1 (Type system $\mathcal{T}$). *Let $\mathcal{B}$ is a set of (atomic) types $S_1 \dots S_n$, $n \geq 1$. Type System $\mathcal{T}$ over $\mathcal{B}$ is obtained by the grammar*

$$\begin{array}{lll} T := S & & \textit{primitive type} \\ \quad | \ (T_1 \rightarrow T_2) & & \textit{functional type} \\ \quad | \ (T_1, \dots, T_k) & & \textit{tuple type} \\ \quad | \ (T_1 + \dots + T_k) & & \textit{union type} \end{array}$$

where $S \in \mathcal{B}$.

Presuming the members of $\mathcal{B}$ being non-empty disjoint sets, then the type $(T_1 \rightarrow T_2)$ means a set of all (both total and partial) functions from T_1 into T_2.

[1] In the context of this work, we understand by the term "program" an XML query we aim to evaluate.

$(T_1, \ldots, T_n)$ denotes the Cartesian product $(T_1 \times \ldots \times T_n)$ and $(T_1 + \ldots + T_n)$ denotes the union $T_1 \cup \ldots \cup T_n$. We denote $o : T$ an object o of type T (also called "T-object").

Regular Type System Subsequently, we define a regular type system $\mathcal{T}_{reg}$ that extends the type system $\mathcal{T}$ with regular constructs. For this definition, we employ the set $Bool \equiv \{TRUE, FALSE\}$ with its common Boolean semantics. Next, let us suppose a set $Name$ that contains all possible element names allowed by the XML grammar [1].

Definition 2 (Type System $\mathcal{T}_{reg}$). *Let $\mathcal{B} = \{String, Bool, Name\}$, $\boldsymbol{tag} \in Name$. The type system $\mathcal{T}_{reg}$ over $\mathcal{B}$ is defined as follows.*

$$
\begin{array}{lll}
T := \boldsymbol{tag} : String \mid \boldsymbol{tag} : & & \textit{elementary regular expression} \\
\quad \mid T* & & \textit{zero or more (Kleene closure)} \\
\quad \mid T+ & & \textit{one or more (positive closure)} \\
\quad \mid T? & & \textit{zero or one} \\
\multicolumn{3}{c}{\textit{where } T \textit{ is an alternative or elementary regular expression.}} \\
\quad \mid (T_1 \mid T_2) & & \textit{alternative}
\end{array}
$$

Upon this type system we can define a type system for XML denoted $\mathcal{T}_E$ as follows.

$\mathcal{T}_E$ — The Type System for XML The last step for obtaining a type system suitable for modeling XML data is a slight alternation of $\mathcal{T}_{reg}$ aiming to support data types available from DTD. For this purpose, we have to take a closer look at DTD properties. Due to the fact that we consider only typed XML documents (i.e. always with an attached DTD) we can distinguish the type of each particular XML element in a document. It is also obvious that in a document there can exist two different elements with the same tag and content, therefore we have to be able to treat them as distinct instances. Therefore, terms *abstract element* and *element object* (or *T-object*) were introduced.

Definition 3 (Abstract Element). *For each XML element there exists exactly one* abstract element. *The (infinite) set of abstract elements is denoted as E.*

Definition 4 (Element Object). *An element object of type T, denoted as T-object, is a partial function of type $E \rightarrow T_{rng}$ where T_{rng} is an* element type *or String.*

An arbitrary XML element is then modeled as a mapping from one particular *abstract element* to its respective codomain (whose type is determined by corresponding *element type*). Hence, by application of an T-object to this *abstract element* we may obtain either a character string (typed as `PCDATA` in DTD) or a more complex structure regarding to the type system.

Definition 5 (Type System $\mathcal{T}_E$). *Let $\mathcal{T}_{reg}$ over $\mathcal{B}$ be the type system from Definition 2 and E be the set of abstract elements. Then the type system $\mathcal{T}_E$ induced by $\mathcal{T}_{reg}$ is defined as follows.*

$$
\begin{array}{lll}
E := \boldsymbol{TAG} : T \mid \boldsymbol{TAG} : & & \textit{elementary element types} \\
\quad \textit{where } \boldsymbol{tag} : T \textit{ and } \boldsymbol{tag} : & & \textit{are elementary regular expressions over } \mathcal{B}. \\
\mid E* & & \textit{zero or more (Kleene closure)} \\
\mid E+ & & \textit{one or more (positive closure)} \\
\mid E? & & \textit{zero or one} \\
\mid (E_1 \mid E_2) & & \textit{alternative} \\
\mid \boldsymbol{TAG} : (E_1, \ldots, E_k) & & \\
\quad \textit{where } \boldsymbol{tag} \in Name & & \textit{complex element types}
\end{array}
$$

Elementary element types and complex element types are hereafter denoted as *element types*. For example, with respect to the DTD shown in Figure 2, we write respective *element types* as:

```
BIB : BOOK*
BOOK : (TITLE, (AUTHOR+ | EDITOR+), PUBLISHER, PRICE)
AUTHOR : (LAST, FIRST)
EDITOR : (LAST, FIRST, AFFILIATION)
TITLE : String
LAST : String
FIRST : String
AFFILIATION : String
PUBLISHER : String
PRICE : String
```

Due to constraints given by DTD, there exists exactly one content model for each XML element (see the "local tree languages" in [11]). Therefore, we can omit the part beyond the colon because it cannot lead to any misunderstanding; i.e., we may use a shorter notation and write `AUTHOR` instead of `AUTHOR : (LAST, FIRST)`.

For convenience, we also define a *nullary function*. It is a useful construct that allows us to access abstract elements and filter them by corresponding element type. Later on, we will utilize it in the query language.

Definition 6 (Nullary function). *A T-nullary function returns the domain of a T-object, where T denotes an element type.*

Note that the set E of all abstract elements is thus split by these functions into a number of disjoint subsets for all respective *element types*.

Attributes. Until now we have discussed only elements, their structural properties, and their content. Other important data in XML are elements' attributes. In the XML-λ Framework we use the same way for accessing them as for elements; we model attributes as functions too. For example, we model the `year`

attribute of the `book` element as a partial function $YEAR : E \to String$ whereas its domain is $E_{BOOK} \subseteq E$ such that $BOOK$-nullary function is there defined (thus, this function is defined only for elements that have associated attributes with this name).

2.3 Data Model Definition

A *data model* is an abstract model that describes how data is represented and accessed. In our work we employ simple and functional types defined by the $\mathcal{T}_E$ type system (see Section 2.2). Apparently, in the XML-λ Framework the main entity to be described is the XML document. We can see the instance of an XML document (which is basically a valuation of a type from $\mathcal{T}_E$) as a structure containing

- set of abstract elements, denoted E_{doc} (subset of the infinite set E),
- set of T-objects,
- set of all text content of document's elements and attributes.

Note that for simplification we consider character strings typed as *String* for all textual content instead of more accurate definitions of `PCDATA` and `CDATA` proposed in [1].

A Data Model Example This section provides an example of a type system definition and realization of an XML document instance in the XML-λ Framework. Let us consider a sample DTD[2] shown in Figure 2 and a respective XML document in Figure 3.

```
<!ELEMENT bib  (book*)>
<!ELEMENT book  (title,  (author+ | editor+), publisher, price)>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT author  (last, first)>
<!ELEMENT editor  (last, first, affiliation)>
<!ELEMENT title  (#PCDATA)>
<!ELEMENT last  (#PCDATA)>
<!ELEMENT first  (#PCDATA)>
<!ELEMENT affiliation  (#PCDATA)>
<!ELEMENT publisher  (#PCDATA)>
<!ELEMENT price  (#PCDATA)>
```

Fig. 2. An example XML schema — `biblio.dtd`

[2] This DTD is taken from the "XMP" Use Case published in the XML Query Use Cases [3] specification

```
<?xml version="1.0"?>
<!DOCTYPE bib SYSTEM "biblio.dtd">
<bib>
  <book year="2008">
    <title> XML technologie </title>
    <author>
      <last> Pokorny </last>
      <first> Jaroslav </first>
    </author>
    <author>
      <last> Richta </last>
      <first> Karel </first>
    </author>
    <publisher> Grada </publisher>
    <price> 286.00 </price>
  </book>
</bib>
```

Fig. 3. A well-formed and valid XML document

For given schema we obtain (as illustrated in Section 5) following set of both elementary and complex element types: $\{BIB, BOOK, AUTHOR, EDITOR, TITLE, LAST, FIRST, AFFILIATION, PUBLISHER, PRICE\}$. The set of abstract elements, E_{doc}, contains for the XML document in Figure 3 eleven items: $E_{doc} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}\}$. The set of element objects with corresponding mappings is illustrated in Table 1.

In this scenario, for instance, the *title*-element object (a function of type $E \rightarrow String$) is defined exactly for one abstract element (the one that serves for modeling the `title` element) and for this abstract element it returns value "XML technologie".

In a more complex case, function *book* of type $BOOK : E \rightarrow E \times 2^E \times E \times E$, applied to an arbitrary abstract element from its domain, returns a quadruple — subset of this Cartesian product. Within the tuple, we can then access its particular components by performing name-based projections (more precisely, "element type name"-based projections).

3 Framework Features Summary

The previous section describes formally the XML-λ Framework along with a simple example. In this part of the contribution, we discuss various aspects of the Framework, its attributes, features, weaknesses, and drawbacks. We have selected nine topics that are, as for us, important for comprehension, usage, and comparison of existing data models for XML. We present such brief comparison with the XML Infoset [4] and the XQuery 1.0 and XPath 2.0 Data Model [7].

Function	Function domain/range	Mappings
bib	$BIB : E \to 2^E$	$bib(e_1) = \{e_2\}$
book	$BOOK : E \to E \times 2^E \times E \times E$	$book(e_2) = (e_3, \{e_4, e_7\}, e_{10}, e_{11})$
book_year	$YEAR : E \to String$	$book\_year(e_2) =$ "2008"
title	$TITLE : E \to String$	$title(e_3) =$ "$XML\ technologie$"
author	$AUTHOR : E \to E \times E$	$author(e_4) = (e_5, e_6)$, $author(e_7) = (e_8, e_9)$
editor	$EDITOR : E \to E \times E \times E$	none
last	$LAST : E \to String$	$last(e_5) =$ "$Pokorny$", $last(e_8) =$ "$Richta$"
first	$FIRST : E \to String$	$first(e_6) =$ "$Jaroslav$", $first(e_9) =$ "$Karel$"
affiliation	$AFFILIATION : E \to String$	none
publisher	$PUBLISHER : E \to String$	$publisher(e_{10}) =$ "$Grada$"
price	$PRICE : E \to String$	$price(e_{11}) =$ "286.00"

Table 1. Set of T-objects induced by the running example

Following paragraphs discuss each topic one-by-one and this text is later summarized[3] in Table 2.

1.Functional Approach. One of the main forces at the rise of XML-λ was the aim to provide an alternative to existing W3C proposal. Therefore the Framework is strictly tied to the concept of functions; it uses both a functional type system and a query language based on simply typed lambda calculus.

2.Multiple Data Models. The XML-λ Framework is presented here as a tool for modeling and querying (and updating) XML formatted data typed by the DTD. Considering the properties of the relational model we can also provide a relational type system within the Framework and work with relational data, or even pursue heterogeneous data transformation and integration between XML and relational data sources.

In addition, although the DTD is still a relatively sufficient solution for most real XML applications we claim that we can provide a type system that is based on W3C's XML Schema [5]. This extension of the Framework will be available in our consecutive work. Yet there are (at least for now) some formal difficulties and therefore we expect that we will be able to support only a part of the specification.

3.Full XML Coverage. Within the Framework we can access and modify only elements and attributes. With regard to the XML Infoset Data Model [4] there may occur other information items in an XML document — e.g., comments, processing instructions, etc. At the present time we do not plan to extend this model but it is certainly possible to enrich it with other sorts of such items.

[3] Y/N/? stands for Yes/No/Unknown; feature **is** supported, **is not** supported, or is **not examined** yet.

4.Uniform Data Model Access. The XML-λ data model does not strongly distinguish between elements and attributes. We model them with the same approach — by functions; and access their content by evaluating these functions independently on their "origin".

5.Element Ordering. Formal foundations of the Framework stand on the set theory. For its utilization in the world of XML we have to employ a kind of ordering among elements. The concept known as "document order" [6, Section 2.4] is realized in the Framework as a partial function $f : E \rightarrow Integer$ that assigns to each abstract element an unique number according to this specification.

6.Mixed Content Model. This is probably the weakest spot of the Framework. With respect to proposed type system it cannot handle data with mixed content model [1]. This is still an issue that must be resolved as soon as possible since it disables the usage of the Framework for document-centric XML data.

7.Mandatory Typing. Another questionable feature is the necessity of an XML schema existence for all XML data processed by the XML-λ Framework. It is a natural requirement with respect to the fact that the type system is built upon this schema. Moreover, for vast majority of XML data available on the web such schema does exist and is used [10].

8.Support for Recursive Types. XML data may also contain some recursive content, even though in small amount only [10]. However, all three data models we mention here can settle this situation.

9.Performance. This aspect is one of the most important data model characteristics for its production deployment. We have carried out some preliminary XPath 1.0 benchmarks with promising results in [14] but we consider this early evaluation as superficial and do not publish and discuss details here. With no doubt, it is one of important issues that have to be furthermore examined.

Feature	**XML-λ**	**Infoset**	**XDM**
1. Functional Approach	Y	N	N
2. Multiple Data Models	Y	N	N
3. Full XML Coverage	N	Y	Y
4. Uniform Data Model Access	Y	N	N
5. Element Ordering	Y	N	Y
6. Mixed Content Model	N	Y	Y
7. Mandatory Typing	Y	N	N
8. Support for Recursive Types	Y	Y	Y
9. Performance	?	?	?

Table 2. The comparison of selected data models

4 Prototype Implementation Overview

We already have a Java prototype implementation available. The main part is a core library that realizes the type system and data model. Existing applications, an XPath 1.0 Processor and the ExDB [9] database management system, are built upon this library. The internal structure of this component is illustrated in the UML class diagram in Figure 4.

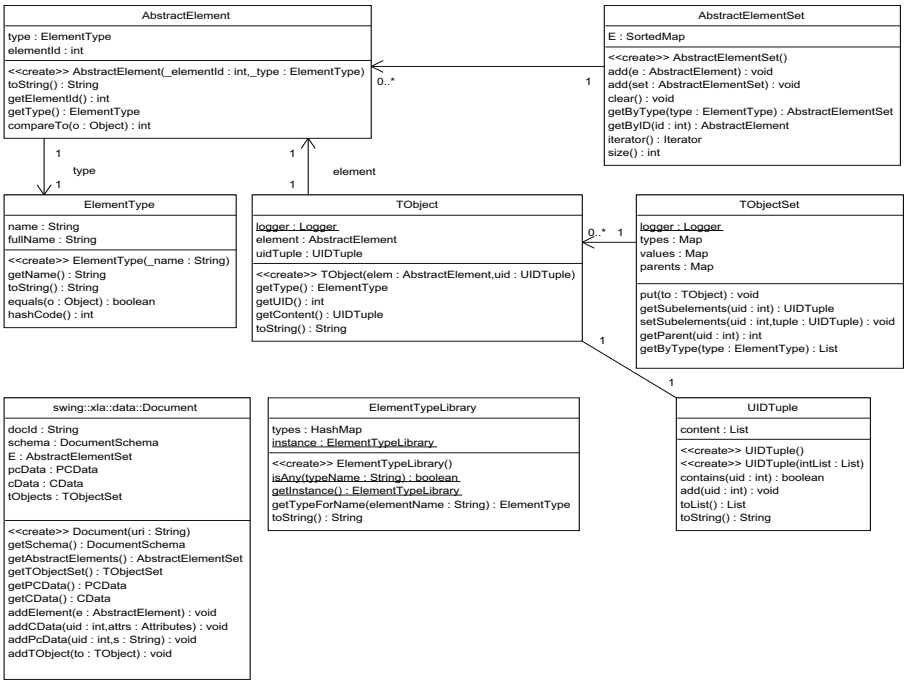

Fig. 4. The main classes in the `swing.xla.core` package

The core package comprises of approximately 20 classes that provide basic functionality as a SAX handler, `XMLOutputFormatter` and other classes that cover the essential functionality.

Our preliminary benchmark of the XPath 1.0 processor based upon this library is ready to be published in [14]. This work contains a comparison with state-of-the-art XPath processors inspired by the XPathMark test [8]. We consider the results as promising but the test did detect a serious performance leak in XML document parsing; in contrast to the others, the process of memory allocation and internal data model structures building behaves exponentially instead of linearly as expected. It is an implementation issue that must be fixed in order to catch up with the rivals.

5 Conclusion

We have published a detailed description of the XML-λ's type system based on the Document Type Definition along with a data model suitable for description of XML formatted data. This model is subsequently demonstrated on a simple example where we describe the concept and utilization of the Framework. Further, we have discussed various issues and features of our approach. We can observe that some of the features are interesting for more detailed examination, e.g. the elegance of the functional approach combined with a reasonable performance of our prototype implementation. Despite of few remaining issues, we still believe that the idea of a functional approach we have presented is interesting, interim results are promising, and our work on this topic will bring meaningful accomplishments.

Future Work. So far, we have proposed a theoretical base for modeling and querying XML. As shown in this paper there are still issues to be solved, in particular the ability to treat mixed element content and developments with the prototype implementation. We find these topics apparently as the most important for the moment.

Acknowledgments. This research has been partially supported by the grant of The Czech Science Foundation (GAČR) No. GA201/09/0990.

References

1. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (fourth edition), August 2006. http://www.w3.org/TR/2006/REC-xml-20060816.
2. L. Cardelli. *Handbook of Computer Science and Engineering*, chapter 103: Type Systems. Digital Equipment Corporation, 1997.
3. D. Chamberlin, P. Fankhauser, D. Florescu, M. Marchiori, and J. Robie. XML Query Use Cases, March 2007. http://www.w3.org/TR/2007/NOTE-xquery-use-cases-20070323/.
4. J. Cowan and R. Tobin. XML information set (second edition), April 2004. http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.
5. D. C. Fallside, P. Walmsley, H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, P. V. Biron, and A. Malhotra. XML Schema 1.0, October 2004. http://www.w3.org/XML/Schema.
6. M. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 Data Model, September 2005. http://www.w3.org/TR/xpath-datamodel/.
7. M. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 Data Model, January 2007. http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/.
8. M. Franceschet. XPathMark: An XPath Benchmark for the XMark Generated Data. In S. Bressan, S. Ceri, E. Hunt, Z. G. Ives, Z. Bellahsene, M. Rys, and R. Unland, editors, *XSym*, volume 3671 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 2005.

9. P. Loupal. Experimental DataBase (ExDB) Project Homepage. http://swing.felk.cvut.cz/~loupalp.
10. I. Mlýnková, K. Toman, and J. Pokorný. Statistical Analysis of Real XML Data Collections. In *COMAD '06: Proceedings of the 13th International Conference on Management of Data*, pages 20 – 31. Tata McGraw-Hill Publishing Co. Ltd., December 2006.
11. M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4):660–704, 2005.
12. J. Pokorný. XML functionally. In B. C. Desai, Y. Kioki, and M. Toyama, editors, *Proceedings of IDEAS2000*, pages 266–274. IEEE Comp. Society, 2000.
13. J. Pokorný. XML-λ: an extendible framework for manipulating XML data. In *Proceedings of BIS 2002*, pages 160–168, Poznan, 2002.
14. J. Stoklasa and P. Loupal. Benchmarking a lambda calculus based XPath processor. Yet unpublished.
15. The World Wide Web Consortium. W3C Homepage. http://www.w3.org.
16. J. Zlatuška. *Lambda-kalkul*. Masarykova univerzita, Brno, Česká republika, 1993.