Database Engineering from the Category Theory Viewpoint

David Toth

Department of Computer Science and Engineering
Faculty of Electrical Engineering, Czech Technical University
Karlovo náměstí 13, 121 35 Praha 2, Czech Republic
tothd1@fel.cvut.cz

Abstract. This paper gives an overview of XML formal models, summarizes database engineering practices, problems and their evolution. We focus on categorical aspects of XML formal models. Many formal models such as XML Data Model, XQuery Data Model or Algebra for XML can be described in terms of category theory. This kind of description allows to consider generic properties of these formalisms, e.g. expressive power, optimization, reduction or translation between them, among others. These properties are rather crucial to comparison of different XML formal models and to consequent decision which formal system should be used to solve a concrete problem. This work aim is to be the basis for further research in the area of XML formal models where category theory is applied.

1 Introduction

In this paper we will focus on some peculiarities from today's database world. Now, in spring 2008, we have many database technologies, many technical frameworks, many solutions for different and similar problems. What we do not have is a global point of view of databases (DB); theoretical approach stating theorems about database models and languages. This paper summarizes database technologies from higher perspective and introduces some of the terms from mathematical category theory (CT). These two aspects, databases and category theory, are put together in order to give new look at the database technologies, to give new way of data model and languages description; and to find new language in which we could ask and answer more generic questions, e.g. about expressive power of (query) languages of particular data models.

This paper deals with databases. More precisely we should say it treats problems which appear when we would like to know which database technology should be used in software project. There are generaly more requirements leading one to use DB, e.g. to make the data persistent, to assure concurrency, etc. More about database technology in general can be found in Date's *Introduction to Database Systems* [13]. There are many factors influencing the decision. In fact in praxis it is more subjective (personal or team) decision.

V. Snášel, K. Richta, J. Pokorný (Eds.): Dateso 2008, pp. 37–48, ISBN 978-80-248-1746-0.

From the software engineering point of view there are at least these kinds of factors; (1) *Human* factors, as e.g.: knowledge about particular DB product, concrete DB technology experience; individual or team, subjectivelly favourite / preferred DB technology; (2) *Technical* aspects: vendor influence, e.g. offered support, problem solving time, programming languages support, performance, accessibility, clustering, and many others. (3) *Problem definition*: teoretical aspects of problem, data itself, its nature (data character), i.e. (a) structuralization (no inner structure e.g. streams, files respectively, weak structure e.g. newspaper articles, strong structure any well structured forms, e.g. tax return form), (b) data contain metadata (typical for XML documents), data separated from metadata respectively (typical for tables—relations). (4) Possibly *other aspects*.

Some of these factors are summarized in SWEBOK [39]. In software engineering paper we would like to address especially the first two categories. In SIGSOFT [36] and especially in SEN [34] can be found more on these topics. But this text is intended to be considered as more database-oriented. Therefore we will focus more on the problem's aspects as the third factor mentioned above. Nevertheless all topics covered here are closely related to software engineering and even to database engineering which we deal with later. Next we will take a closer look at particular database technologies emphasizing the problem's aspect.

1.1 Relational Database Technology

Historically the first database approach which solved the inconsistencies, redundancy, concurrency and other problems was the relational model. C. J. Date in his *Introduction* [13] deals with the relational approach to represent data (i.e. relational data modeling and storing among other aspects). Other very deep insight into relational data model can be found in E. F. Codd's *Relational model for database management* [10]. One can imagine the main idea as data grasped via a relation in mathematical notion, i.e. all data can be viewed as relations, in other words sets with internal structure of its elements. Relations are interconnected together using values of particular set of elements which is usually called foregin key usage.

Can any data be represented using relational approach, i.e. can any data be stored as relations, tables respectively? We must consider the fact that the data could possibly change its structure, and even that we do not know the structure before we have the data physically. Can the changing data structure be modeled using relational approach? What other questions play a significant role when we consider expressive power of e.g. relational algebra, etc.? These and related questions will be considered in future works which will contain CT oriented features. In this paper data model description and related topics will be treated. Further we focus on object technologies.

1.2 Object Database Technology

Object and object-oriented databases arose out of the impedance mismatch between relational and object data models. The essence of this problem lies in a

different kind of data representation, i.e. once as relations or n-aries and once as objects. The problem inhere in data translation. In other words data must be mapped between classes of objects and relations of n-aries. More about the object relational mapping can be found in Fussel's *Foundations of Object Relational Mapping* (ORM) [20]. Another paper about ORM can be found in [1]; implementation issues are covered in persistence framework Hibernate [24].

To object and object-oriented databases and to object database management systems is dedicated the web site [32]. At this place many object and object-oriented technologies, especially database technologies, of course, and open source as well, can be found.

1.3 XML Databases

The XML Databases were born actually very shortly after the XML, the new language for semistructred data description, a W3C's standard respectively [17], emerged in 1998. More about XML evolution can be found at [46]. It did not took a long time and new term Native XML Database, often just NXD, came abroad [37]. We will use the term, as e.g. R. P. Bourret in [7] does. Why NXD appeared and what to expect from them is described in [23] or [6]. R. P. Bourret [8] also maintain fresh list of NXD products and even more wide XML products in general.

The main motivation for NXD usage resides in the impedance problem again, as in case of ODB as well. The typical situation where NXD are used is web portals and web applications communicating through web services. The web services standards are based on XML and related standards. Therefore it is evident that the need for XML document transformation should be avoided to speed up the performance of applications of this type. We have treated this yet earlier in [42].

The principle of NXD consists in XML data model as intrinsic data model of the database engine. R.P. Bourret is more specific about what XML-Enabled and what XML-native suppose to mean, e.g. in [7].

The paper is structured as follows. Section 2 deals with relationship of software engineering and database engineering. Section 2.2 refers problems relevant to appropriate database technology selection in the introduction above bearing in mind. Section 3 summarizes issues related to essences of particular data models. In section 4 there are mostly XML related standards and technical part of XML databases dealt with and section 5 treats formalisms developped for the purpose of XML Data model description. Section 6 introduce basic terms from category theory (CT) and gives formal background for cited formalisms. Section 7 summarizes exhibited XML formalism. Last section 8 reveals our future plans.

2 Software Engineering and Database Engineering

Terminological note on database engineering. Normally the term database engineering is used to describe an area of processes, methods and techniques,

formalisms and languages useful for database designing, and in general, useful for database application development. There are several conferences around database engineering topics. Clear definition of what is database engineering does not exist. What we mean under database engineering is specialization of software engineering practices for purposes of database application, i.e. application strongly related to data which makes persistent and which further operates with. Practically we mean specialization of all techniques where arbitrary database artifact, most typically it is database schema, is created, changed (most common case), or removed.

From waterfall to iterative development. Software engineering in past was understood as sequential processes equivalent to phases which must be performed in a serial way. The phases typically are: feasibility study, business analysis, requirements analysis, architecture analysis and design, logical design, GUI design, DB design, physical design, coding, testing, refactoring, installing, deploying, measuring, among others. The same can be said, as an analogy of course, about database engineering, i.e. database design, database tuning and administration among others. But in today's world when agile methodologies in sofware engineering become successfull and more and more widespread, it also seems to be inevitable to use agile or generally speaking iterative approaches in database community. It is a paintful step for every single database expert long time experienced sequential approach when starting use agile principles.

MDA—Model Driven Architecture. The MDA approach [30] seems to be contradictory in the context of agile methodologies. But not necessarilly. Iterative approaches allow to build software systems more focused on one particular problem, emphasizing one aim in time (during iteration). The basic imagination could be as very little waterfalls chaining every iteration stressing analysis or design or programming depending on current phase. We can figure out here the semantics of the word phase depends strictly on chosen methodology.

2.1 Database Engineering and Evolutionary Approach

From the point of view of the database engineering there is need to elaborate database design. Typically conceptual model is considered as a part of database modeling and as a part of a database design phase. In fact we would like to stress here that there is no need to create domain model as UML [31] class diagram during business analysis and also E-R diagram as a part of database modeling independently. It is possible to create or even generate E-R diagram or UML class diagram in Data Modeling Profile from domain model. It is typically expressed as UML class diagram which is done during business analysis. Actually some CASE tools offer this functionality in these days, e.g. the Enterprise Architect [15].

Database modeling, a part of database design, can be viewed as a transformation from domain model. And this does not mean that all the modeling must

be finished before normalization or tuning starts. The core of the evolutionary approach lies in doing the whole step by step in very small parts which have to be integrated. Continuous refinement is necessary. One of the biggest argument against iterative database development is the need for neverending reworking and refining of non-stabilized artifacts—which is possibly a great number.

As a resume here we would like to pinpoint the possibility to look at database evolution concurrently with regular software evolution. And therefore to see database engineering as a specialization of software engineering. The principles of MDA—model transformations are essentially the same in software and database engineering. This kind of abstraction should help us thinking in software engineering and database engineering in very similar way. Furthermore CT can help us when dealing with models, their properties and qualities, and transformations.

2.2 The Database Technology Selection

Which particular DB technology should we choose to use? What should lead us — help us? The discussion below involves these questions.

Relational Databases (RDB). From the historical perspective there is a big argument which says to use RDBs. It is deep insight into relational technology, strong mathematical background in form of data relational model and relational algebra. Many people made refinements of this technology for a long time. Shortly, RDBs are greatly elaborated in comparison to other (and younger) technologies.

Object Databases (ODB). In [32] we could find at least these important reasons why to select ODB instead of RDB or XDB: embedded DBMS application, complex data relationships, deep object structures, changing data structures, development team is using agile techniques, massive use of object oriented programming language, there are many objects including collections, data is accessed by navigation rather than query.

One of the most popular ODBMS in open source community is db4objects [14]. Another example could be the NeoDatis ODB [29] or GemStone/S [21].

XML Databases (XDB). With XDB, and NXD respectively, fine-grained reuse of content is possible; NXD allows sophisticated hypertext applications with mixture of stuctural and fulltext query. The most typically cited NXD benefits are flexibility and reuse.

We have treated of this issue in greater detail in [41]. Three distinct metrics, ρ, τ, and ξ, were proposed for different kinds of database technologies.

3 Essentials of Data Models

Does exist essential difference between different data models? In words of CT we could say: belong all categories of all data models into the same category (of categories)? We will focus a bit more on this in section 6 — The Category Theory Standpoint.

Now imagine not to use CT. The question if there exists any problem which cannot be solved using arbitrary technology would have to be proven hardly. We would have to prove that every single case of data expressed in one data model could also be expressed in every other data model.

Theoretically any data can be expressed in arbitrary format, i.e. (1) tables, nested tables respectively, (2) the web of objects or (3) hierarchy of elements if we found mappings between all data instances.

Mapping from XDB to RDB can be viewed so that any XML document can be stored (represented) in RDB in generic tables (ELEMENTS, ATTRIBUTES, DOCUMENTS, etc.). That objects can be stored as record in tables which can be seen e.g. in Object Relational Mapping (ORM) Pattern. The other way can be imaginated as direct overwriting of RDB data using wrapping method for column content and nesting in case of foreign keys (FK). FK can also be represented as ID and IDREF attributes in XML documents.

Mapping from XML documents to objects can be grasped in a way that XML data model will be grasped as a tree, object model would be accessed as a graph. A tree is also a kind of a graph. This idea is demonstrated e.g. in previous work [42], and it is implemented in java programming language in JAXB—Java API for XML Binding [38]. These mappings are typically based on DTDs or XML Schema or even RelaxNG. R. P. Bourret wrote general paper on XML document mapping between relational and object models [5].

The same could be obtained if we find all the mappings between one and another DB structure — technology (RDB, ODB, XDB). But a more convenient way would be to find out the way of general description and prove that these mappings have to exist or that it is impossible these mappings would exist. And not only convenient, we should consider all data models; even those which do not exist yet. It seems that different technologies fit for different kind of problems but they are essentially the same after all. Are they? Can we prove this using category theory? We would like to focus our future research on these questions.

And there are other interesting questions leading us to finding one framework only, CT, e.g. is it possible to store and effectively retrieve data with unknown and/or changing data structure in RDB, ODB and XDB?

4 XML Databases

XML standard [17] first released in 1998 and last updated in 2006 has initiated the great interest in XML Databases and NXDs.

R. P. Bourret summarizes and yet reconciles not only basic problems and principles of native XML databases in [7]. In his article R. P. Bourret says: "...

the problem is practical, not teoretical ..." about the problem of arbitrary data expressed in any data model. He also states "... in RDB there is an impractical number of joins ..." in [6].

Another resource concluding the benefits of NXD usage [23] tries to list not only the advantages but also the possible problems.

We will very shortly summarize here XML database technologies and in the next section we will cover formal models for technological standards and data models treated here. Three typical NXD representants are as follows: (1) One of the most common NXD's is eXist [16]. (2) Another very popular NXD from Apache is called Xindice [45]. Oracle Berkeley XML DB is described at [33].

XML formal models and languages from the point of view of XML standard are at least as follows. We could say the following list is an extension of XML Data Models according to R. P. Bourret [7]. The majority of all treated models are tree-based formal models and algebras.

- DOM — Document Object Model [40].
- SAX — Simple API for XML [28].
- Infoset [11].
- XPath [9].
- XQuery [18].
- XML-λ: functional approach to XML description [27].
- Most likely there are many other standard-based formalisms.

Next section reveals the formal background of stated standards and needed relationships.

5 XML Databases Formal Models

A Formal Data Model and Algebra for XML [3] is the name of the article suggesting a tree-based model as a formal data model for XML and as an algebra for XML, an algebra based on such trees, i.e. essentially same structure as DOM and the related.

XML Data Model as it is defined in XPath or XQuery is basically grasped as a forrest of trees of nodes representing elements and attributes and texts and other XML features mentioned in previous section. Many of the XML Data Model facets are explained in XML infoset [11].

As an XML data model could be grasped DOM. Basically the formalism is build on same terms as in case of XPath or XQuery. So from the CT point of view it would be grasped as one formalism.

Sengupta and Mohan summarized in [35] the formalisms used to describe data in XML format. They found these formalisms:

- Tree-based formalisms (XAlgebra, DOM and others).
- SAL — Semi-structured Algebra [4].
- The ENF — Element Normal Form concept: it is proved that attributes can be avoided in cases of general description because every XML document with attributes can be (without any information loss) transformed onto the XML document variant without attributes and vice versa.

- HNR — Heterogeneous Nested Relations — also arise from NF^2 (Non-first normal form).
- HNRC — HNR Calculus — analogously to relational calculus.
- HNRA — HNR Algebra — analogously to relational algebra.
- DSQL — Document SQL — as an analogy to SQL.

For all of these we would like to find the proper meta-formal way of description in terms of CT; and finally find out the properties valid among these categories.

XML Algebra based on monads is another interesting formalism [19]. But this approach, this XML Algebra, lacks references and dereferences. The algebra specified in [3] count on it and offer a way of how to solve this problem.

P. Wadler proposed several formal models. Especially formal semantics for XSL [43], and semantics for XPath [44].

Future challenges would be to describe formalisms used for metamodels—conceptual models and visualisations e.g. via UML.

Having in mind the extent of all this we will focus on just few factors from the previous list in CT. We introduce CT in the next section.

6 The Category Theory Standpoint

This section deals with an introduction to CT and categorical description of XML formal models defined above. Let us take a look at the word category itself.

6.1 Three semantics of the word Category

Categories originally arose in mathematics out of the need of a formalism to describe the transformation from one type of mathematical structure to another. Category represents a kind of mathematics. Barr and Wells [2] state then category as a mathematical workspace.

A category is also a mathematical structure. It is then a generalization of both ordered sets and monoids. Barr and Wells call it in this case category as a mathematical structure.

Category as a theory is the third recognized point of view. Category can be seen as a structure that formalizes a mathematican's description of a type of structure. Traditional way to do this in mathematics, in mathematical logic respectively, is to use formal languages with rules, terms, axioms and equations.

We now define the term category more precisely. We will use the notation and mathematical formalism used in [2] for the rest of this section.

6.2 Definition of a Category

Definition 1. A category $\mathcal{C}$ consists of **objects** (denoted by $A, B, C, ...$) and **morphisms** between them (denoted by $f : A \to B, g : B \to C, ...$). These data

are subject to obvious axioms expressing **composition**, its **associativity**, and existence of **identity** morphisms (units w.r.t. composition).

A paradigm category is the category $\mathcal{S}et$ of all sets and mappings. See [2] for more details.

6.3 Definition of CCC, Connections to λ-Calculus

We define now the concept of a cartesian closed category (CCC). It is proved in [26] that CCC's are *essentially the same thing* as simply typed λ-calculus. Altough the following definition is rather a technical, one may bear in mind that the category $\mathcal{S}et$ forms a paradigm example of a CCC.

Definition 2. A category $\mathcal{C}$ is called a cartesian closed category (CCC) if it satisfies the following:

(1) There is a terminal object 1.

(2) Each pair of objects A and B of C has a product $A \times B$ with projections

$$p_1 : A \times B \to A \text{ and } p_2 : A \times B \to B.$$

(3) For every pair of objects A and B, there is an object $[A \to B]$ and an arrow $eval : [A \to B] \times A \to B$ with the property that for any arrow $f : C \times A \to B$, there is a unique arrow $\lambda f : C \to [A \to B]$ such that the composite

$$C \times A \xrightarrow{\lambda f \times A} [A \to B] \times A \xrightarrow{eval} B$$

is f.

Note that we call an object 1 of a category $\mathcal{C}$ terminal iff there is exactly one arrow $A \to 1$ for each object A of $\mathcal{C}$.

λ-calculus is one of the formal description of what is usually called an XML data model. In [27], there is XML-λ approach to view XML. The typical point of view of XML is a tree. But this approach emphasizes the notion of a function. And there is a hypothesis that as functions or as trees we describe the same, and that both kinds of description are of the same power. We will try to prove this in future work. This proof will rely on what is stated above.

Related works involving Object Databases description using CT are [22] and [25]. Altough it is not about the XML data model the principles of formal description are very similar.

Because of the λ-calculus is one of the formal description or precise point of view of XML data model and because of what Lambek and Scott proved in their work [26], the XML data model can be described as CCC. We would like to find out if also other categories as descriptions of other formal models are also CCC. The main idea is to determine if all the models are also essentially the same in the sense of Lambek and Scott; which should be done in next work.

6.4 Proposed Descriptions based on Category Theory

The very first description we considered was the XML-λ approach. This approach is an instance of the λ-calculus theory which, grasped as a category, is CCC [26].

Let G be a graph. As a graph we mean special case of oriented graph with loops on nodes. The category $\mathcal{C}_{Graph}$ of such graphs is defined as follows: Collection of objects consists of all possible graphs G; Collection of arrows consists of all graph homomorphisms ϕ_G. Identity arrows are isomorphisms of objects. It is needed to be verified, that this mathematical structure is a category, but it is obvious; we let this to the kind reader. Furthermore this category is CCC, as is proved e.g. in [2]. This model, category $\mathcal{C}_{Graph}$, is actually a useful model for object databases [25] when other aspects than object visibility are ignored. Objects in this category can be grasped as objects from object programming. But as a model for XML databases it cannot be used because of the loops. When the arrow is interpreted as a relation of nesting, element in XML document cannot be nested into itself.

Let $\mathcal{C}_{Tree}$ be the category of trees (derived from the category above). Let objects be trees and arrows tree homomorphisms. Again that it is a category is needed to be verified as above. This category is not CCC. Because there would needed to exist the terminal object with loop node. But such an object cannot be interpreted as any XML document.

Let $\mathcal{C}_{HFS}$ be the category of hereditary finite sets. All these sets can be undrestood as ϵ-trees. This approach seems to be very promising and is currently under development.

There are many other approaches which will be in detail covered in subsequent works.

7 Conclusions

We have shown that the database technology selection is in praxis mostly subjective problem. There are few practical reasons which would lead us to develop theoretical framework for data modeling.

We have stressed the natural evolution in software engineering from waterfall to iterative database evolution approaches which still become more common.

We have discussed XML formal models and their properties.

The conclusions from CT applications are rather poor. But we tried to summarize the database problems, existing solutions and we tried to offer another, originial, approach.

Furthermore this work open the doors for further more specific research, using very strong mathematical background. Next section reveals our future plans.

8 Future Works

Subsequent work will be focused on the question of essentiality of the RDB, ODB and XDB models, their computational equivalence, expressive power of relative languages and similar aspects.

In the near future we will try to categorify every formal model for XML data which would be found.

In far future there is a huge space for using CT formalism to describe itself, i.e. use the notion of categories of categories. And according to Lambek and Scott [26] it seems to be possible use only one notation, one language and one formalism, we mean CT of course, for all (types of) data models. We would like to try to find out such a way of description of data models.

Next, in the future, not only XML databases and NXD will be described using CT. But we would like to try to give formal basis for all data models. Good example could be relational algebra and Crole's way of categorical description which should be further elaborated [12]; using categorical semantics. And there are many other similar examples as an inspiration for future research activities.

References

1. S. W. Ambler. Mapping Objects to Relational Databases: O/R Mapping In Detail. 2006. http://www.agiledata.org/essays/mappingObjects.html.
2. M. Barr and C. Wells. *Category Theory for Computing Science*. International Series in Computer Science. Prentice-Hall, 1990. Second edition, 1995.
3. D. Beech, A. Malhotra, and M. Rys. A formal data model and algebra for XML, 1999.
4. C. Beeri and Y. Tzaban. SAL: An algebra for semistructured data and XML. In *WebDB (Informal Proceedings)*, pages 37–42, 1999.
5. R. P. Bourret. Mapping DTDs to databases, 2001. http://www.xml.com/lpt/a/2001/05/09/dtdtodbs.html.
6. R. P. Bourret. Going native: Making the case for XML databases, 2005. http://www.xml.com/pub/a/2005/03/30/native.html.
7. R. P. Bourret. XML and databases, 2005. http://www.rpbourret.com/xml/XMLAndDatabases.htm.
8. R. P. Bourret. XML database products, 2007. http://www.rpbourret.com/xml/XMLDatabaseProds.htm.
9. D. Chamberlin, A. Berglund, and e. a. Scott Boag. XML Path Language (XPath) 2.0, September 2005. http://www.w3.org/TR/xpath20/.
10. E. F. Codd. *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
11. J. Cowan and R. Tobin. XML information set (second edition), April 2004. http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.
12. R. Crole. *Categories for Types*. Cambridge Mathematical Textbooks. Cambridge University Press, 1993.
13. C. J. Date. *An Introduction to Database Systems*. Addison-Wesley Publishing Co., Inc., 2003. 8th ed.
14. db4objects — Open Source ODBMS. http://www.db4o.com.
15. Sparx's Systems Enterprise Architect UML CASE Tool. http://www.sparxsystems.com.
16. eXist — Open Source Native XML Database, Home Page. http://exist.sourceforge.net.
17. Extensible Markup Language (XML) 1.0 (Fourth Edition), 2006. http://www.w3.org/XML.

18. 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/.
19. M. Fernandez, J. Simeon, and P. Wadler. A semi-monad for semi-structured data. *Lecture Notes in Computer Science*, 1973, 2001.
20. L. M. Fussel. Foundations of Object Relational Mapping. http://www.chimu.com.
21. GemStone/S ODB. http://www.gemstone.com/products/smalltalk.
22. J. Güttner. *Object Databases and the Semantic Web*. PhD thesis, 2004.
23. E. R. Harold. Managing XML data: Native XML databases, 2005. http://www.ibm.com/developerworks/xml/library/x-mxd4.html.
24. Hibernate — Java and .NET persistence framework. http://www.hibernate.org.
25. P. Kolenčík. *Categorical Framework for Object-Oriented Database Model*. PhD thesis, 1998.
26. J. Lambek and P. J. Scott. *Introduction to Higher-Order Categorical Logic*. Cambridge University Press, March 1988.
27. P. Loupal. Querying XML with lambda calculi. In *Ph.D. Workshop, VLDB2006*, 2006.
28. D. Megginson. SAX – Simple API for XML, 2005. http://www.saxproject.org/.
29. NeoDatis ODB. http://wiki.neodatis.org.
30. Object Management Group (OMG). MDA — Model Driven Architecture, 2007. http://www.omg.org/mda.
31. Object Management Group (OMG). UML — Unified Modeling Language, 2007. http://www.uml.org.
32. ODBMS — Object And Object Oriented Database Management Systems. http://www.odbms.org.
33. Oracle Berkeley XML DB, home page. http://www.oracle.com/database/berkeley-db/xml/index.html.
34. SEN — Sigsoft Software Engineering Notes. http://www.sigsoft.org/SEN/surfing.html.
35. A. Sengupta and S. Mohan. Formal and conceptual models for xml structures - the past, present, and future, 2003.
36. SIGSOFT — ACM's Special Interest Group, dedicated to Software Engineering. http://www.sigsoft.org.
37. S. Staken. Introduction to Native XML Databases. 2001. http://www.xml.com/pub/a/2001/10/31/nativexmldb.html.
38. Sun Microsystems, Inc. Java architecture for XML binding (JAXB), 2003. http://java.sun.com/webservices/jaxb/.
39. SWEBOK — Software Engineering Body Of Knowledge. http://www.swebok.org.
40. The W3C Consortium. Document Object Model (DOM), 2005. http://www.w3.org/DOM/.
41. D. Toth and P. Loupal. Metrics analysis for relevant database technology selection. In *Objekty*, 2007.
42. D. Toth and M. Valenta. Using Object And Object-Oriented Technologies for XML-native Database Systems. In J. Pokorný, V. Snášel, and K. Richta, editors, *DATESO*, CEUR Workshop Proceedings. CEUR-WS.org, 2006.
43. P. Wadler. A formal model of pattern matching in XSL. Technical report, 1999.
44. P. Wadler. Two semantics for xpath, 1999.
45. Apache Xindice, Home Page. http://xml.apache.org/xindice/.
46. XML Main Page. http://www.w3.org/XML.