Development Problems in XML Algebraic Parsing Process

Adriana Georgieva
Fac. Applied Mathematics and Informatics
Technical University of Sofia, TU-Sofia
Sofia, Bulgaria
e-mail: adig@tu-sofia.bg

Bozhidar Georgiev
Fac. Computer Systems and Control
Technical University of Sofia, TU-Sofia
Sofia, Bulgaria
e-mail: bgeorgiev@tu-sofia.bg

***Abstract* - In this paper, are presented some problems and solutions concerning the implementation of proposed algebraic method for XML data processing. The proposed theoretical researches and practical realizations lead to faster XML parsing process. Here is suggested a different point of view about the creation of advanced algebraic parser. This point of view is in tight connection with some popular concepts of the functional programming. Therefore, here proposed nontraditional approach for fast XML navigation using algebraic tools contributes to advanced efforts in the making of an easier user-friendly API for XML transformations. This way, the programmer can avoid the difficulties about the complicated language constructions of XSL, XSLT and XPath languages. The choice of programming language C# is a logical consequence, which follows some previous experiments with other high level programming languages. These activities were carried out by the same authors. The discussed specific search mechanism based on the use of algebraic functions is theoretically and practically faster in comparison with many other well-known XML parsers. Finally, the conclusion is that in this area really exists a possibility for creating new software tools, based on the linear algebra theory, which can completely replace the whole XML navigation and search techniques used for the present by XSLT/XPath.**

Keywords - hierarchical XML tree structure, functional programming (FP), XML transformations of semi-structured data, algebraic modeling of XML structures, module-finite algebra, XPath scripting language, XML parser.

I. INTRODUCTION

The main purpose of this paper is the research about the possibilities for application of one nontraditional approach for addressing the components in XML tree. This approach is based on the principles of the functional programming (FP). According to the cited incontrovertible sources [1][12], the functional programming is a specific programming technique tightly connected with function definitions. In practice, the difference between a mathematical function and the notion of a "function" used in imperative programming is that imperative functions can have side effects, changing the value of already calculated computations. That's why, they lack referential transparency i.e., the same language expression can result in different values at different times [12] depending on the state of the executing program. The formal description of BNF (popular as Backus-Naur Form) template concerning functional programming is: *program::= set of functions*. Functional programming finds use in real practice through several programming languages like Mathematica (symbolic math), F# in Microsoft .NET and XSLT (XML). Spreadsheets can also be interpreted as FP languages. Actually, in the theory are presented many different ways concerning function description - tables, equations, definitions, etc. In view of the fact that the paradigm FP is a mathematical abstraction rather than a programming language, we lay particular stress on types of functions, which are widely used at present. Higher-order functions are functions that can either take other functions as arguments or return them as results. Higher-order functions are closely related to first-class functions [2]. Higher-order functions and first-class functions both allow functions as arguments and results of other functions [8][9]. The distinction between the two functions is subtle: "higher-order" describes a mathematical concept of functions that operate on other functions, while "first-class" is a computer science term describing programming language entities that have no restriction on their use (thus first-class functions can appear anywhere in the program, including as arguments to other functions and as their return values).

Actually, higher-order functions enable partial application or currying, a technique, in which a function is applied to its arguments one at a time, with each application returning a new function that accepts the next argument. In other words, functional programming is a style of programming that emphasizes the evaluation of expressions, rather than execution of commands. The widespread use of XML prompted the development of appropriate searching and browsing methods for XML documents [4]. The presented paper offers a particular point of view focusing on the building of an algebraic formalism for navigation over XML hierarchy connected with functional programming theory. With the use of XML query languages, users of XML retrieval systems are able to exploit the structural nature of the data and restrict their search to specific structural elements within an XML documents [3].

Definitely, most paradigms for defining a variety of query languages are based on either way of the two logics widely used in the context of trees – first-order logic (FO), and monadic second-order logic (MSO). MSO extends FO by the quantification and navigation over the sets of nodes. In this paper, we shall consider monadic second-order logic (MSO) as first-order logic extended with "monadic second-order variables" ranging over sets of XML elements. In other words, the query languages for extraction of data from XML documents (XSL, XSLT, etc.) are grounded theoretically on MSO logic. XPath language is rather related to FO logic [5].

This article presents a nontraditional point of view that connects the application of FP principles (as illustration of the declarative style of programming) with here proposed algebraic approach. The purpose of this FP model representation is to make the implementation of advanced linear algebra tools [9] possible for XML data manipulations. In Section I the main functionalities of FP and the basic goals of this paper are shown. Section II describes the conceptual model connected with the proposed algebraic approach for faster search in XML hierarchy. The exhibited in this section results follow previous researches of the authors which have been exposed in [2]. Here are discovered the internal links between some theoretical formulae, which show the possible substitution of complicated language constructions (XPath, XSLT) with the discussed above FP techniques. Section III presents some XML parser architectures and program realizations along with an algebraic search and hierarchy access. Finally, in Section IV the general issues, conclusions, the further researches and some open problems are discussed.

II. ALGEBRAIC APPROACH FOR FASTER SEARCH IN XML HIERARCHY

To avoid the bottleneck, that characterizes the languages XSLT/XPath for XML transformations, there is necessity to accelerate the parser process and node access in common XML hierarchy. This section is dedicated to some possibilities for faster search and navigation over XML hierarchical trees by means of linear algebra tools. The presented formulae can be considered like functions, based on module-finite algebra tools [10]. According to cited researches [8] [9] and as result of proposed theoretical model [7], in this paper is given the unique determination of the physical address of the object O_k^r (object r from level k) in common hierarchical structure i.e., the number p_k^r by the following way:

$$p_k^r = \sum_{i=1}^{k-1} \alpha_i + a_k^I = \alpha_1 + \alpha_2 + \ldots + \alpha_{k-1} + a_k^I$$

$$= \alpha_1.\Phi(h_1) + \alpha_1.\Phi(h_2) + \ldots + \alpha_1.\Phi(h_{k-1}) + a_k^I =$$

$$= \alpha_1.\sum_{i=1}^{k-1} \Phi(h_i) + a_k^I \,, \qquad (1)$$

where: - $\Phi(h_1) = c_0 = 1$;

$\Phi(h_2) = c_0.c_1 = c_1$;

$\Phi(h_3) = c_0.c_1.c_2 = c_1.c_2$; .... ;

$$\Phi(h_k) = c_0.c_1.c_2. \ldots .c_{k-1} = c_1.c_2. \ldots .c_{k-1}$$

are here defined transformed characteristic elements from the tree;

- $c_0, c_1, c_2. \ldots .c_{k-1}$ - the number of children (subordinated elements) of any element from level *i* to level *i+1*; $c_i \in \mathbb{Z}$; ordinary $c_0 = 1$ and therefore:

$$a_k^I = \{\ldots\{(a_1-1).c_1 + a_2 - 1)\}.c_2 + \ldots + (a_{k-1}-1)\}c_{k-1} + a_k - 1 \qquad (2)$$

Here a_k^I is the code value a_k in the hierarchical level *k*. The calculations in formula (2) are based on the formal description of the sets of code values of XML nodes components. According to suggested formal algebraic description [2] each of the objects in a real XML hierarchical data structure can be accepted as an element of the corresponding hierarchical structure [9]. For XML physical data design, in this article is chosen one-dimensional address array with codes of all XML database elements from the type:

$E\,(i_1 \div k_1, i_2 \div k_2, \ldots, i_n \div k_n)$, which presents the XML data structure in increasing consistency in the order of the corresponding hierarchical levels. On this base the address of element *r* from level *q* in common hierarchy can be determinate by:

$$ADR(r,q) = \sum_{m=1}^{q-1} R_m.D_m + r \,, \qquad (3)$$

where: $R_m = (k_{m-1} - i_{m-1} + 1).R_{m-1}$ and $R_1 = 1$.

In other hand:

$$D_m = (k_m - i_m + 1) / (k_{m-1} - i_{m-1} + 1)$$

Here D_m is the number of the subordinate elements of level *m-1* to level *m*. This formula is valid for cases of the "balanced" hierarchical structures, when the number of the subordinate elements to every element of each level to the next one is constant. That's why, in dependence of the concrete user applications of database structures, the formula (3) is a program, realized so that every element from each hierarchical level has a different number of subordinate elements compared to the next one. If we denote with $a_1^m, a_2^m, ..., a_{\alpha m}^m$ the code value of the elements from level *m* in common hierarchy, then the expression $k_m - i_m$ is a dimension of level *m* in hierarchy for each *m = 1, 2,3 ... ,n.* This algebraic approach allows comparatively simple search of XML hierarchical data by means of the following types of functions – specification functions and nesting functions. As it was shown in [9], the specification functions comprise three basic manipulations for data handling: specification manipulation on only one level, structural specification manipulation that returns all lower levels and quantity specification level – returns all possible levels in horizontal and vertical order. As can be seen in these specification functions, the existent relationships between the elements of the different hierarchical levels are mainly from two types: relations between elements within the structure – *inside-structure relations* and relations between elements of the different hierarchical structures – *inter-structure relations.* Most of these relationships are either from type "one-to-one" or from type "one-to-many".

Let we consider two basic hierarchical structures **T** and **P** in one XML document and corresponding relations between elements of them.

Definition 2.1. The relations between elements from the same hierarchical structure representing relationships between them, as in the one hierarchy - structure **T (**or in other hierarchy- structure **P)** we call relations of strict order and strict inclusion [7][10].

More complicated are the relations from the type "one-to-many" that present the relationships between elements of some hierarchical levels in the **P**-structure – for example the relations between elements of every couple of levels K and L. For these relations is defined the operation "projection" (**pr**) for each element from K to L.

Definition 2.2. Each element $k_i \in K$ correlates with non-empty set of elements $\{l_j\} \in L$, which we call "intersection" by k_i and will denote with $\mathbf{r}(k_i, l_j)$.

The intersections by k_i i.e., given element is a set of such subjects $\{l_j\}$, that $(k_i, l_j) \in \mathbf{r}$, where i=1, …,*n* ; j=1, …,*m* (*n* is a number of elements of K, *m* is a number of elements of L and it is not obligatory $i \neq j$). For example, when:

K=$\{k_1, k_2, k_3, k_4\}$, L=$\{l_1, l_2, l_3\}$ and

$$\mathbf{r}\{(k_1, l_1), (k_1, l_3), (k_2, l_1), (k_2, l_2), (k_3, l_2), (k_3, l_3), (k_4, l_4)\}$$

then $\mathbf{r}[k_1] = \{l_1, l_3\}$; $\mathbf{r}[k_2] = \{l_1, l_2\}$; $\mathbf{r}[k_3] = \{l_2, l_3\}$; $\mathbf{r}[k_4] = \{l_3\}$,

moreover always exist at least one $\mathrm{r}[k_i] \neq \varnothing$, because projection $\mathbf{pr}(r(k_i, \_)) \neq \varnothing$ as, for example, $\mathrm{pr}(r(k_i, \_)) = \{l_1, l_3\} \neq \varnothing$ … , etc.

These intersections by $k_1, ..., k_n \in K$ present the specific peculiarities of relation **r** between elements of levels K and L in hierarchical structure **P.** Similarly, here can be described the relations between other couple of hierarchical levels from this type in any XML document.

III. HOW THE PROGRAM SYSTEM (ALGEBRAIC PARSER) IS BUILT?

For the purposes of presented research is assumed that the physical records in the XML hierarchical file are with fixed length. On Fig. 3.1 is depicted the functionality of proposed in this article XML parser. Usually, XML documents are stored in physical memory of computer by means of a variety of index-sequential methods. Each element from a given level in the common hierarchy includes different number of siblings and child elements. It means that any object O_n^r is represented with the code value (integer) p_n^r, which is defined from the disposition of the element in XML hierarchy. Here *n* is the number of hierarchical level in common structure and *r* is the place number in the fixed level from left to right. This algebraic processor supports a code table with the names of elements of current XML document and the corresponding integers p_n^r , which uniquely define the place of the object (node) O_n^r from level *n* in the real hierarchical structure i.e., its address in the physical XML database.

These algebraic presentations of the binary relationships in hierarchical structures remove the necessity from table's work, relation schemes, etc. So it operates only with rows of numbers, which leads to use of ordinary algebraic tools for data transformation from XML structures to their presentation on physical level.

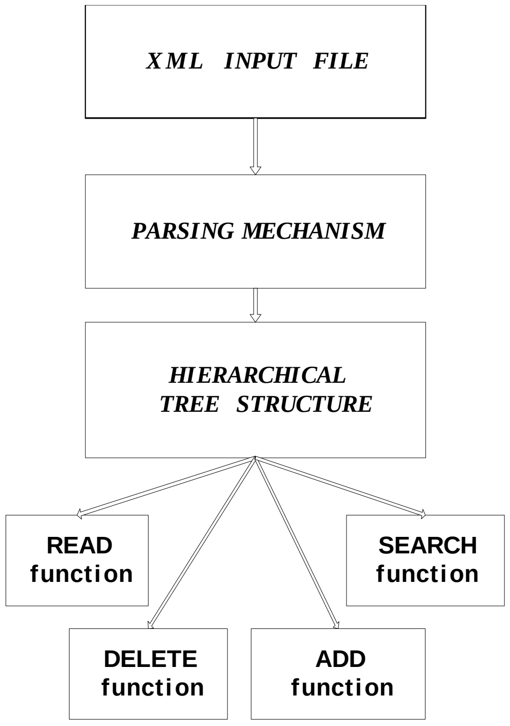

Figure 3.1. Common scheme of functional XML parser

The file information after the parsing processing is saved in the same way as it is kept in the XML input hierarchy. Search function gives the oportunity to the user for tag searching procedure. Final results show the content of the corresponding tag and its position in XML hierarchy. The algebraic approach makes obvious the possibilities for reaching linear time functions in XML tree hierarchy handling. The user can insert an additional information in XML file according to previously defined input format. Some operations for common use could be done as follows:

- DELETE operation of element a_k^r - in this case a_k^r =0 means removal of this element from the table along with its descendants down in the hierarchy until to the last level *n*.

- GET operation uses the formula (1) for immediate address search of an assigned in advance element.

- INSERT operation puts in the table the name of element and its coordinates; here is necessary to increase the index of other elements on the right side of the element, for example a_k^{m+1}, etc.

For more flexibility and best user convenience there is foreseen the possibility that provides WEB access to the parser. The presented diagram on Fig. 3.3. describes in details classes, used in project realization.

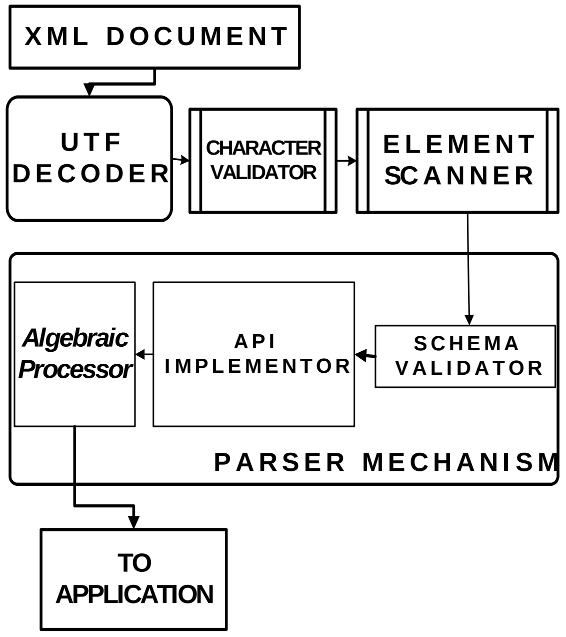

Figure 3.2. XML Parser Architecture added to the algebraic searching and hierarchical tree access

On Fig. 3.2 is defined a set of normative functions for use with the proposed in this paper processor.

For the program realization of XML parser here is used MS Visual Studio environment. Visual Studio [11] is chosen for this purpose in view of the fact that this programming package provides wide spectrum of software tools, used to develop both console applications and WEB applications (and services) as well. Visual Studio supports different versions of programming languages C/C++, VB.NET, C#, XML/XSLT, HTML/XHTML, JavaScript and CSS. Other popular languages can be easly supplemented to this MS program environment after simple instalation. The following elements of MS Visual Studio are used in the building process of XML parser:

- Code editor which facilitates text colors and recognizes variables, functions, methods and other components through its core module IntelliSense.
- Debugger module is implemented for tracing programing code about errors detection and correction in input file.
- Designer modules: Windows Forms Designer for creating Windows Forms graphical interface and Web designer/development for WEB sites aplications.

The first practical realization of so proposed algebraic parser uses programming tool Eclipse SDK for Java [9]. This article is dedicated to the acceleration process of this parsing mechanism using language C#.

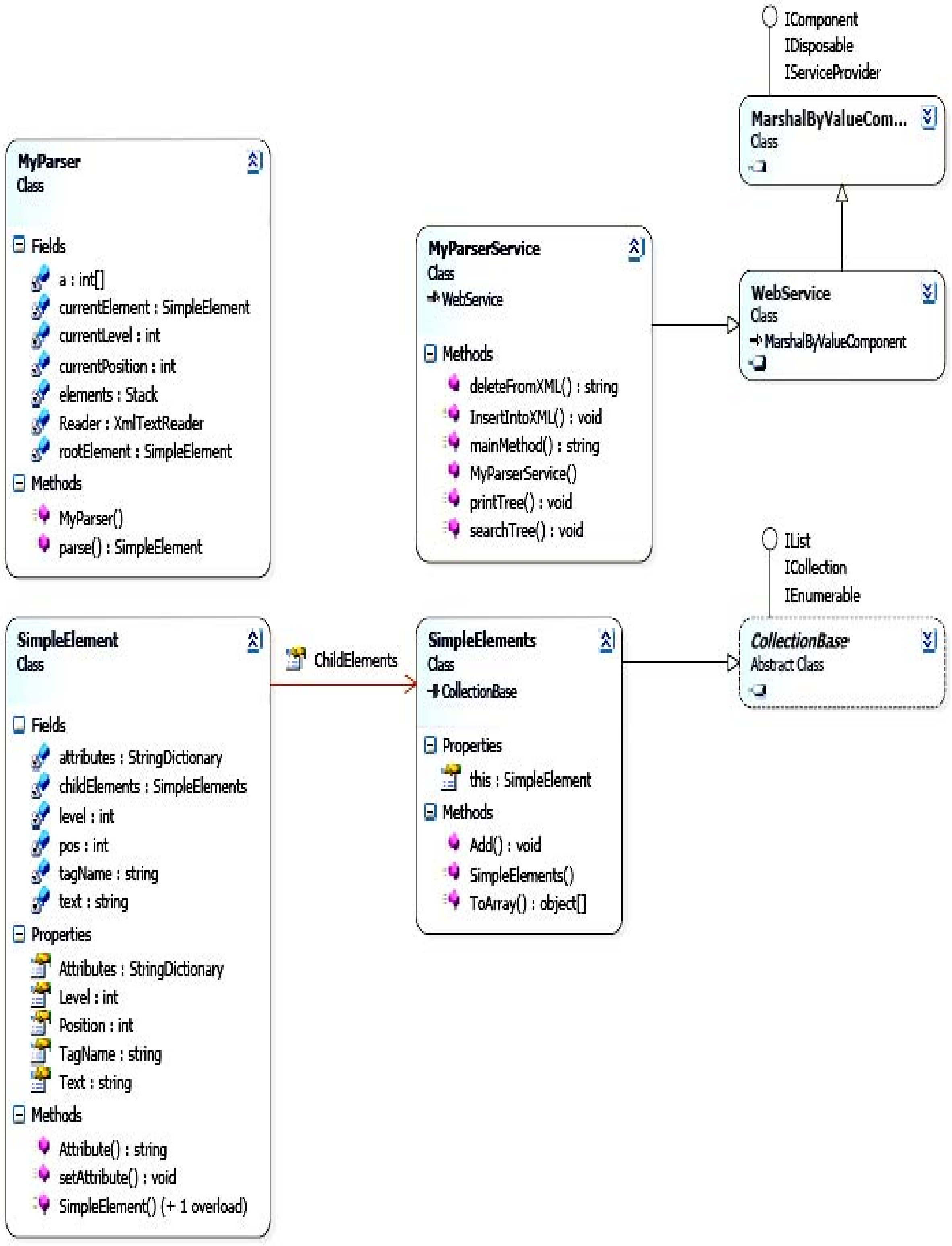

Figure 3.3. Functional class diagram of the parser

The results of both practical researches (JAVA and C# realizations) demonstrate faster accomplishment of discussed XML operations in comparison to some traditional approaches, especially to these which are based on the search in XML hierarchy (XSLT/XPath).

IV. CONCLUSION AND FUTURE WORK

This article is an author's attempt to create a mechanism for accelerating XML document processing, connected with the main principles of FP. The presented XML parser is built and practically works by using advanced algebraic formulae. The authors reveal several basic algebraic operations, which are included in proposed in this article parser and make logical connections with some concepts of the functional programming. On the basis of this model, that presents the addresses of elements in XML hierarchical document as integer values, is possible to work with common algebraic mechanism. This mechanism is used in modeling of relationships between the different XML hierarchical components, data operations and standard specification functions [6]. It allows a possibility to work with ordinary linear operators in harmony with FP theory instead of the more difficult operations, which are specific for the widely used models. The algebraic approach points out some opportunities for search and navigation in different WS*-specifications especially as WSDL, BPEL, UDDI, etc. Here are suggested algebraic mechanisms for advanced algebraic processor creation (with all necessary programming modules). This nontraditional (functional) approach about the faster navigation with the presented algebraic tools promotes to build new interface for XML search techniques and transformations. The implementation of so proposed method in the area of SOA could accelerate the various types of XML message-based communications concerned WSDL, UDDI, BPEL, SOAP, etc. This algebraic mechanism can be dynamically located, invoked and combined with other navigation techniques. It gives users the opportunity to process their data messages easier and faster, that is of critical importance to make service-oriented paradigm operational in the real practical environment. Finally, the proposed approach is different in comparison with many other well known query and transformational languages in respect of their definition, expressiveness and search techniques. Several research questions remain as it is mentioned below:

- The results of this paper justify a natural fixed point in the development of some future possibilities for matrix presentation of the relationships between the elements of the different hierarchical levels in XML hierarchy (inside structure and inter-structure relations). Actually, using matrices will be very convenient for conceptual representation and programming realization of hierarchical relationships between every two levels in the whole data structure.

- The presented paper offers an algebraic point of view for building of algebraic processor (with appropriate software and programming techniques). Therefore, this particular point of view about an algebraic processor based on the tools of linear algebra and motivated from FP theory, conducts to software results, which eliminate the complicated language constructions of XSL, XSLT and XPath.

REFERENCES

[1]. J. Darlington, P. Henderson, and D. A. Turner, Functional programming and its applications, Cambridge university press, ISBN 0 521 24503 6, 1982

[2]. A.Georgieva and B. Georgiev, "A Navigation over XML Documents through Linear Algebra Tools", The Fourth International Conference on Internet and Web Applications and Services - ICIW 09, Venice/Mestre, Italy, 24-29 May 2009, Published by IEEE Computer Society, ISBN: 978-0-7695-3613-2/09.

[3]. J. Keogh and K. Davidson , XML DeMYSTiFied, McGraw-Hill, Emeryville, California, USA, 2005.

[4].World Wide Web Consortium, http://www.w3.org/TR/2004/. Extensible Markup Language (XML) 1.0. W3C Recommendation, third edition, February 2004.

[5]. L. Libkin, "Logics for unranked trees: an overview", Department of Computer Science, University of Toronto, 2006.

[6]. B. Georgiev, "An Approach for Some Implementations of W3C-Standard XML", Second International Scientific Conference, Kavalla, Greece, 2005.

[7]. A. Georgieva,"Algebraic Modelling of Hierarchical Data Structures on Conceptual Level", Proceedings of Second International Scientific Conference "Computer Science'2005, Chalkidiki, Greece, Part II, pp.113-118.

[8].A. Georgieva and B. Georgiev," Conceptual Method for Extension of Data Processing Possibilities in XML Hierarchy", Forth International Scientific Conference, Kavalla, Greece, 2008.

[9]. B.Georgiev and A.Georgieva, "Realization of Algebraic Processor for XML Documents Processing", AIP Conference Proceedings of 36-th International Conference AMEE-2010, vol.1293, pp. 279-286.

[10]. L. Garding and T. Tambour, "Algebra for Computer Science", Spring-Verlag, N.Y., 1988.

[11]. J.Sharp, Microsoft Visual C# Step by Step, Microsoft Press, Washington, 2008

[12]. P. Hudak, "Conception, Evolution, and Application of Functional Programming Languages", ACM Computing Surveys 21/3, 1989, pp. 359-411.