Parameter Centric XML Parsing Techniques for Embedded Systems

Rashmi Sonar*, Sadique Ali** and Amol Bhagat***
*Department of Computer Science and Engineering,
Prof Ram Meghe College of Engineering and Management, Badnera, Amravati, India
rashmi.sonar@gmail.com
**Department of Computer Science and Engineering,
Prof Ram Meghe College of Engineering and Management, Badnera, Amravati, India
softalis@gmail.com
***Innovation and Entrepreneurship Development Center,
Prof Ram Meghe College of Engineering and Management, Badnera, Amravati, India
amol.bhagat84@gmail.com

Abstract: Embedded systems are dedicated to special purpose so they are different than conventional system. XML is known standard for communication because of the features like scalability, openness, flexibility. XML must be processed efficiently when it is used as a communication language. Parsing is one of the major tasks in processing XML. XML parsing itself is the challenge to achieve especially when the recourses are limited and tasks are with deadline. XML parsing in embedded systems has dedicated purpose which is vital in accomplishment of required tasks. In this paper different parameters for XML parsing are analyzed. The parameters for enhancing the efficiency of real time embedded systems are presented in this paper. This paper discusses the various XML parsing techniques available for embedded systems. The presented three basic approaches of parsing can be used for devising new approaches for XML parsing in embedded systems. As evaluated using experiments, VTD can handle large size XML documents. Therefore VTD can be considered for development of XML parsers in real time embedded systems.

Keywords: Document object model, embedded systems, parameter based parsing, real time systems, simple API for XML, virtual token descriptor, XML parsing.

Introduction

XML processing consists of four stages namely parsing, access, modification, and serialization. The parsing consists of three-steps. First step is character conversion where xml document character bit sequence is converted into character set of host programming language. Second step is lexical analysis in which character stream is converted into tokens like start element, attribute, end element. Final step is syntactic analysis where actual well formedness of document is verified. The character conversion and lexical analysis are invariant through any parsing model. The syntactic analysis represents data for access and modification with application of different parsing models. Some basic models are document object model (DOM), simple API for XML (SAX), and virtual token descriptor (VTD). DOM creates tree based API to represent XML document. SAX creates event based API with push and pull based parsing respectively, and VTD parsing method parses the XML document and creates 64-bit binary format VTD record (token) for each event [1].

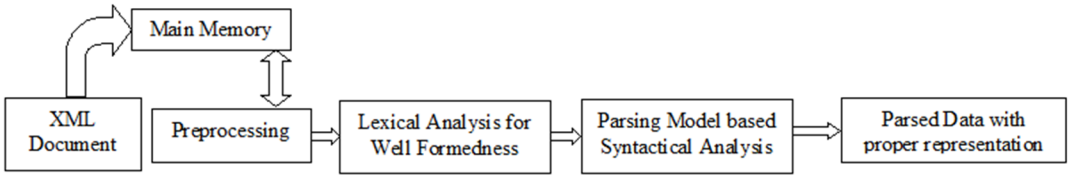

Figure 1. XML parsing process in embedded system

XML is de-facto standard for communication in message passing from one desktop to other. Many researches are there for desktop performance. XML web services is the most used technology for realizing service oriented architecture (SOA) because of the features like easy use, compose-ability, modularity, support, low cost, and commonality. For optimum utilization of SOA, XML behavior in web services also plays important role [2]. The XML parser implementation in

embedded systems differs from conventional computing systems primarily for the limited resources on embedded systems. Embedded systems are dedicated for particular purpose so XML parser should also be designed for accomplishing that purpose. In embedded systems primary requirement is to transfer XML syntax to internal format such as c [3, 4]. The overall architecture of XML parser in embedded system is shown in figure 1. Here pre-parsing is an important stage. As the embedded system having the limited memory, there is necessity to manage the data such a way that the better results must be produced efficiently. The processing time of XML comprised of parsing time and preprocessing time. In this paper section 2 compare different approaches of XML parsing for real time embedded systems, section 3 presents identified parameters for enhancing the performance of XML parsing in embedded system, section 4 evaluates three basic parsing techniques which can be extended in real time embedded system environment.

Related Work

In performance evaluation of XML web services for real-time applications, [2] defines use of SOA and XML web services in real time business application and system with the example of two banking scenario. For the integration of different types of software application, service oriented architecture (SOA) is one of the methods. The XML web services evaluation is performed so that each adopter can get SOA for large and complex system [2]. For high performance web services when typical schema is required, an adaptive XML parser, based on table driven XML (TDX) technology, plays an important role [5]. In this parser scanning, parsing and validation are performed in one pass with tabular representation of schemas and push down automaton at runtime. The parser achieves high performance without reconstruction and redeployment of web services which is five times greater than traditional parser.

A schema specific parser [6], combines parsing and validation into a single pass and achieves better response time. The technique used is table driven streaming XML parser in tabular form. For TDX, a toolkit is constructed for developing parser in C which comparable with DFA based parser. When asynchronous message passing required, a non-blocking schema specific parsing method for XML with two stack push down automaton [7], without backtracking gives the better performance. Parallel processing is also the performance enhancement approach. In [8] a DFA based approach is identified which recognizes XML specification and converts DFA to NFA for any of the input. The parser is based on parallel processing of the large scale xml documents with performance scale as 6-7 cores. Identifying the XML limitation, a cache parser is presented in [9], with the goal as to cache XML document information. Like DOM, cache parser exploits the syntactical tree of an XML document. It is strictly based on sender receiver and takes the advantage of xml document syntactic tree stored in cache.

XML DOM parsers memory characteristics are identified in [10] such as heap composition, object size, and type distribution, object life with JikesRVM2.4.1 and Merlin algorithm. With the proper garbage collector method, the parsing performance can be improved. BNF tree based BNFParser is proposed in [11] in which XML formal grammar is represented by extended Backus Naur form. BNFParser parser carries the idea with run time adaptive and code size efficient parsing which can be suitable for embedded system where storage restriction and memory limitations is the main issue. The only drawback is that, it is applicable to XML data less than 100 KB. Many times the structure of XML document is similar and only the values of attribute changes. The concept of structural encoding is introduced in [12]. Exploring identical XML structure, processing time can be reduced. Coupling is loose when algorithm is less aware of interfaces of parser and generally java is less efficient than C and C++. How to loosely couple the algorithm is analyzed in [13]. In [14], processing rules generator for Java phone is described with use of techniques like the DTD parser, XSLT, Java. This type of processing rule if applied to mobile database [24], mobile data can be read as manipulated.

XML Parsing Performance Enhancement in Embedded Systems

As the embedded systems are dedicated to particular task, there is some typical type of document processing which is having same structure, here values of attribute changes with same skeleton of document. The key point for parsing is no need to parse the whole document but storing the skeleton of XML, only identify values of attribute. In [15] XML structural data is reduced to format strings and arguments are sent as they are generated using modifications of real-time compression techniques specific to each data type. The portions of the XML structure which are common to many packets are generated on the fly or a priori and the values which vary from packet to packet are compressed using techniques specific to the type of data being sent. In this approach correlated and uncorrelated numeric data and short and long text strings are used. Format strings generation for each type of packet is carried out. The format string expresses the structure of the XML data in the packet and the portions which differ from packet to packet (arguments) become all that must be transmitted for subsequent packets. For example, assume a target tracking application generated the following two data packets for a target's location at separate times:

<target><lat>45</lat><lon>50</lon></target>
<target><lat>43</lat><lon>55</lon></target>

The format string could be expressed as <target><lat>[arg1]</lat><lon>[arg2]</lon></target> and the wireless device could just send the arguments after the format string was established. New entries were compressed in the same manner as long strings and the index positions were sent with the minimum number of bits required. Format strings are simply the element structure of the XML packet with the escape characters. If written well, this will be optimal and allow for the highest compressibility; however this would require more training than many users may want to do. While insert messages are simply replace messages with a zero length and delete messages are replace messages with an empty fragment. In [15] comparison of the TinyPack XML against Deflation, XMill, XMLPPM, and PAQ over the four datasets in both delay tolerant and real time experiments measuring compression, latency, processor usage, RAM requirements, and energy consumption is presented. All the data was collected prior to compression and compression was done on the entire dataset at once. TinyPack XML slightly outperforming XMill and XMLPPM and slightly underperforming the expensive "ideal" PAQ algorithm. The SNAResult and track data contained more static structure than the other datasets and required less RAM for TinyPack since the static portions of the structure are only stored in one place and are only compressed once.

The SCBXP technique architecture [16] makes use of a content-addressable memory that must be configured with a skeleton of the XML document being parsed, a finite state machine that controls FIFOs; in order to align XML data properly, multiple state machines acting on the multilevel nature of XML, and dual-port memory modules. In SCBXP technique the production of well-aligned data, at the end of the matching stage, is naturally accompanied with well-formedness checking and validation, without the need to rescan and reprocess the same XML document in terms of well formedness and validation. Prior to processing a new XML document, the SCBXP must perform the task of configuring the CAM with a skeleton derived from the XML document to be parsed. Due to the structure of the skeleton, a successful match of a tagged XML string against any of the CAM contents implies that this string is well formed and validated.

A new method for designing and implementing a manual XML parser named BNFParser is presented in [17], which is based on the mechanism of XML document matching against a BNF tree that built on XML formal grammar represented by extended Backus-Naur form (EBNF) notation. BNFParser is parsing an XML document with size less than 100 KB BNF tree, which embodies all of BNF generations, is composed of three kinds of BNF nodes: non-terminal symbol node, terminal symbol node and symbol group node, and one BNF expression is represented by one or more BNF nodes. The parsing process of BNFParser is matching XML document input against BNF Nodes in BNF tree. The main steps are: during the process, BNF stack is used to keep the latest status of traveling XML BNF tree to assist matching and possible backtracking actions. Compared with YACC/LEX-assisted parsers, BNFParser can work immediately without recompiling when changing XML syntax sets.

Most structure-related processing is identical for data items with identical structure; it is thus evident that the overall performance of XML processing will improve if redundancy in structure related processing can be reduced [18]. Structure encoding and the approaches to quickly identifying recurring structures, including one relying on collision-resistant hash function is introduced in [18]. The techniques to improve the performance of XML transmission, tokenization, parsing, and transformation by using structure encoding is also described in [18]. The implementation of prototypes of structure encoding based XML parser and transformer is carried out by extending the kXML parser and the XT XSLT processor. In structure encoding, the structure of an XML document is derived from the serialized text of the document, after removing non-whitespace text nodes and after "canonicalizing" element tags. "Canonicalizing" element tags includes "whitespace canonicalization" and "attribute canonicalization". "Whitespace canonicalization" removes any optional whitespaces in element start tags and end tags, and replaces any required whitespace with a single space character. "Attribute canonicalization" removes '=' character, the attribute value, and any delimiters surrounding attribute value. Efficient transmission of XML documents is particularly important in mobile environments. Structure recurrence can be exploited for compression and efficient transmission, by avoiding the redundant transmission of structural in formation and by compressing recurring test nodes and attribute values. Implementation does not support documents with variable-length arrays such as lists of identically structured elements with non-fixed lengths.

A conventional serial XML parsing architecture is described in [19] along with an improvement of architecture, using speculative pipeline structure to parsing XML, and speedup the parsing rate. The validator will examine: the validity of particular element or attribute in the document; the element under consideration is a child element; the order and number of child elements at any particular level of hierarchy in the document; the contents of the elements and attributes conform to the specified data-type; the contents of the elements and attributes conform to the specified valid range of values; a particular element or attribute is an empty element or can include text, and the value is default/fixed value. The data flow of XML parsing contains: first, data are loaded from either network or local hard disk. Then, data flow into the memory subsystem: main memory, L2 and L1 caches [20]. At the end, the processor fetches data from cache and performs the actual computation.

The performance bottleneck of XML parsing is in the memory data loading stage, rather than the disk data loading stage or the network exchange stage. In other words, the overhead introduced by the memory subsystem really plays on the weakness of the XML data parsing. Therefore, to speed up the XML parsing execution, it is imperative to focus less on acceleration and instead reduce the overhead incurred by the memory subsystem loading an XML document into memory and reading it prior

to parsing may take even longer than the actual parsing time. Consequently, instead of optimizing the specific computation of parsing, acceleration from the memory side; that is to say, accelerate the XML data loading stage memory-side accelerators deliver considerable effectiveness across existing parsing models is explored. They are able to reduce cache misses by up to 80 percent, which translates into up to 20 percent of performance improvement [20]. For code generation and parsing of XML schema, the open source software EXIficient is used in [21]. EXIficient is the Java implementation of W3C EXI specification. It is able to convert XML files using XML Schema to EXI streams and vice versa. EXIficient XML Schema parser and grammar builder is used to leverage VHDL code generation. EXIficient was extended with additional classes that use the internal representation of EXI structure to generate VHDL code. Then, the VHDL is used for hardware synthesis.

Table 1. Various parameters analyzed for performance improvement of XML parsing in embedded systems

Data Size	Token Descriptor	Memory Size	Parsing Time	Throughput
480 Byte [14]	Binary representation of the information and the structure of the document	-	0.066sec	
Tinypack compression [15]	structure of the XML data in one packet		1.4% of time for processing packet	
CAM is Configured with a Single Skeleton- 15 character or 15228bytes CAM is Configured with a New Skeleton for Each Loaded XML Document 15 character or 15228 bytes [16]	Token Id			25MHz-449Mbps 25MHz-400Mbps
<100 Kb on embedded system [17]	BNF Node	Heap memory <200 KB		
Large Documents, 30.40MB [18]	VTD array and MED Structure	2 times the size of XML document		35 MB/sec
With increment of EXI stream length [19]	EXI (Embedded XML Encoding) Stream		Hardware EXI get 0-10 μs	
Blooming filter length1024, [22]	ordered labeled trees in which the nodes represent elements or values, and the edges represent a relationship between two nodes.	false positive rate estimation p.=0.25 then 150 Kbytes. i.e $5 \times 10{-}5$ times the amount of space	Matching Time 32μs	
< 666KB [23]		PSDXP with 2 threads uses 5% Slice Register, 9% Slice LUT and 8% Block RAM. PSDXP with 4 threads uses 11% Slice Register, 19% Slice LUT and 17% Block RAM.		Percentage of post well-formed checking under overall CPB is 0.0868% and 0.1949% in PSDXPx2 and PSDXPx4 individually. PSDXP can achieve 0.5004 CPB and 0.2505 CPB with two threads and four threads individually.

The models do not explicitly deal with namespace processing, but instead use attribute processing to mimic namespace value processing in [22]. The models are based on the parameters: M- the number of blocks. A block is a structure in the XML document that consists of a parent tag delimited by a start and end tags. Several child elements also delimited by start and end tags, are contained within the parent tag. n- the number of child elements in a block. w- the number of attribute in the parent element of each block. z- the number characters in each text element in a block. x- the number of attributes in each child element in a block. Length (child_name)- the length of a child element name(<address>, length is 7 characters). It can be considered as a function f (child_name) = Length (child). Following the preceded convention f (text)- the length of a text node. f (att_name)- the length of attribute name. f (att_value)- the length of the attribute value. f (block_name)- the length of the block element. Pd- the number of characters corresponding to decorating tags which are the XML processing instructions and the root element.

Table 2. Comparison of available XML parsers

Parser	Implementation Language	Usage	Remarks
LibroXml	C	The library **libroxml** for XML parsing inside applications. The binary **roxml**, an xpath resolver that can be used from shell. The module **fuse.XML** that can be used to mount an XML file as a file system	library is minimum, easy-to-use
PugXML	C	The PugXML parser performs string scanning, tokenization, parsing, and construction of the document tree structure in a single pass.	Presented is a small, fast, non-validating DOM XML parser, contained in a single header.
RomXML	RomXML AE Parsing and Framing toolkit with C-language structures	Translate XML syntax to and from embedded internal C-language structures	More resources available for application features
ElectricXML	Java-based XML parser	Designed to have a small memory footprint and an intuitive operation as part of the GLUE distributed computing platform.	Electric XML parses DTDs but does not perform validation or implement the DOM. The ability to parse SOAP messages significantly faster than popular DOM-based parsers was a significant design goal
MinML	Java	MinML is an XML parser written in Java which implements nearly all of the XML language (it ignores DTDs).	It was developed for use in small embedded systems and has a code footprint of less than 10Kb.
MinML-RPC	Java	MinML-RPC 0.1 is a minimal eXtensible Markup Language Remote Procedure Call (XML-RPC) implementation that will run on small embedded systems (about 512Kb of RAM).	
NanoXML	Java	NanoXML is a very small (5KB) XML parser for Java.	SAX is a SAX adapter for NanoXML
Expat XML parse	C, Expat is an XML parser library written in C.		It is a stream oriented parser that requires setting handlers to deal with the structure that the parser discovers in the document.
TinyXML	C++	TinyXML is a simple, small, minimal, C++ XML parser that can be easily integrating into other programs.	It reads XML and creates C++ objects representing the XML document. The objects can be manipulated, changed, and saved again as XML.
SAX	Java	You can process the XML document in a linear fashion from the top down	
RapidXML	C++	RapidXml is an attempt to create the fastest XML parser possible, while retaining usability, portability and reasonable W3C compatibility	RapidXml achieves its speed through use of several techniques.

Compact structure representation is proposed in [23] using bloom filter, which also provides an easy solution for separation of the parsing process from the matching process so as to relief the burden of parsing from the matching time is significantly reduced due to the separation of parsing and matching, and the space for indexing structure is tremendously reduced due to the compactness of bloom filter. In [23] two issues are addressed. First, the parsing and matching are tightly bound together and cannot be easily separated. As the parsing takes a tremendous amount of time, the time efficiency is severely compromised. Second, the complex indexing and matching algorithms adopted by these works impose the risk of memory (or

storage) space overuse. The preprocessing module is comprised of a parser and a bloom-filter creator. It will be used to process the raw XML documents and XPath filters before the filtering process. They can be embedded at the client side. In this way the parsing processes are distributed over the individual clients themselves, providing a naturally balanced distribution of the computation throughout the system. When a subscription/notification has been generated, it will be first processed by a preprocessing module, which is comprised of an XPath/XML parser and the bloom-filter generator. A bloom filter is created after the processing for each subscription/XML document. The bloom filter combined with a subscriber identifier will be sent to the filtering engine, then indexed and stored for the filtering process. As to the notification, the XML documents will be sent together with the corresponding bloom filters. Once a document has arrived, its bloom-filter will be used for the evaluation against the subscription bloom filters stored previously. The un-subscription process is a simple instruction to remove the record in the filter index, identical to that in a conventional pub-sub system. When a subscription/notification has been generated, it will be first processed by a preprocessing module, which is comprised of an XPath/XML parser and the bloom-filter generator.

Generic XML parser [25] can be used to parse and reconfigure any valid XML file. The application of this parser is particularly useful in Software Communication Architecture where XML files represent the properties and parameters of hardware components and devices. A change in the parameters of the XML files changes the behavior of the hardware components. By using this Generic Parser, the XML files can be reconfigured dynamically at the middleware level and hence control the behavior of the hardware through software. This parser, when used at the middleware level to parse the XML files allows the user application to focus on the application logic itself, without dwelling on the tedious details of parsing the XML. The generic parser creates an info object for each node in the XML file and the structure of the info object is then referred by the DTD or schema. If the input XML file does not have schema, is not correct at the semantics level. The parsing technique used is the recursive descent parsing. A recursive descent parsing is a top down parsing technique where parsing starts from the parent node and proceeds down till it reaches the innermost child node.

After getting the root element of the XML document instance (topmost parent), a hash table is created. A hash table or hash map is a data structure that uses a hash function to map identifying values, known as keys, to their associated values. Thus, a hash table implements an associative array. The hashmap is used to store the key-value pair for all the elements of the instance of the XML document. Once the hashmap is created, the element is traversed till the end to find the attribute list. If an attribute is found, it is added to the attribute list hashmap in the form of (name, value) pair. All the available attributes for an element is added to the hashmap for each element.

All the child nodes are obtained along with their properties starting from the parent node. For this, the child is first checked if it is an instance of the parent node. If it is an instance of the parent element, then the name of the child is obtained. The attributes of the child node are added to a newly created hashmap. This child is treated as a new parent and it is parsed similar to the root node. If it does not have any instance, the name and the value associated with the child node is parsed and printed. A counter is used and hence it is possible to jump to the next child node from the root element directly without having to traverse through the visited nodes all over again.

The design is based on Document Object Model) to alter or change the parameters of the hardware devices through the parser. DOM is a tree-based interface that models an XML document as a tree of various nodes such as elements, attributes, texts, comments, entities, and so on. A DOM parser maps an XML document into such a tree rooted at a Document node, upon which the application can search for nodes, read their information, and update the contents of the nodes. Once it is found, the new value may be entered and it is updated in the document instance. Suitable error handling has been implemented to ensure that the algorithm does not deviate from its original flow. After the updation is completed, the entire document is written back to the original XML file and the updations are reflected in the XML file.

Evaluation of Extensive Parsing Techniques for Real Time Embedded Systems

XML parsers can be classified as heavy and light processors. Processors used for high computation called as heavy parsers which are not suitable for limited memory resulting in inconvenient for embedded systems where limited memory is available. Processors used for limited memory space and limited processing power are called as light processor. Some heavy parses are JDOM, JAXP, Xerces, or Xalan and most of them offer support for both DOM and SAX. Some light parsers are NanoXML, kXML, Xparse-J, ASXMLP, WoodStox, and TinyXML. The DOM and SAX parsers are generally used in conventional platforms. Embedded implementations of DOM and SAX are also available in C and Java. DOM's requirement for storing tree structure is high as compare to SAX's requirement with simple operation and little buffer space is more suitable for embedded system. MinML and NanoXML are examples of embedded parser. VTD-XML is also one of the basic XML parser which is not the object oriented and enables random access of document with the feature to run the application. VTD- XML can be implemented in software or hardware implementation with low resource usage. Though the light weight processor is available, problem of memory fragmentation is there which creates garbage collection in the environment like Java and C# with runtime overhead. Table 2 shows some of the available XML parsers with their usage, advantages and languages in which they are implemented. Any XML Parsing approach is the extension of basic approach like SAX, DOM, VTD. Therefore these approaches are evaluated in this paper for giving the directions for designing and developing novel

XML parsers and extending them for real time embedded system environment. Table 3 shows the experimental result for medium size XML document parsing with these basic approaches.

Table 3. Parsing results of three basic parsing approaches for small XML datasets

File Size (Bytes)	Parsing Time Required (milliseconds)		
	SAX	DOM	VTD
1102	63	57	31
5079	93	63	36
8167	94	63	47
120283	219	78	62
135386	250	93	62
156293	359	97	47
1294444	2324	124	70

The graph shown in figure 2 indicates that when the size of XML document is minimum all the approaches requires near about same time. When the size increases, at some points DOM and VTD require the same time. But for the big size xml document, VTD performs better than DOM and SAX. When SAX parser gets XML input, it generates the events. SAX is event-based and stores nothing in memory. SAX access the small part of document at a time. DOM parser converts the XML input to in memory objects. DOM converts the whole XML document in memory as object tree. IN VTD XML input is converted to tokens based on binary encoding specification. Each Virtual Token Descriptor record is a 64-bit integer that encodes the token length, starting offset, type, and nesting depth of a token in XML.

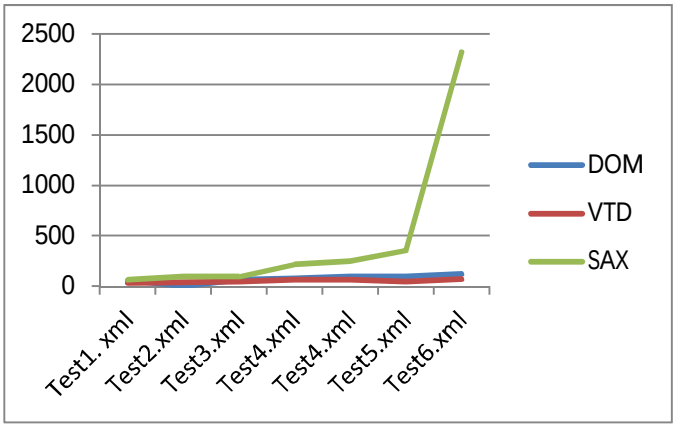

Figure 2. Comparison of SAX, DOM, and VTD on the basis of parsing time

Conclusion

XML parsing is always performance bottleneck. For parsing in embedded system, focus should be on preparsing methods such as reusable objects, garbage collection method, XML document serialization, structure encoding in future. The various performance measures can be concentrated for improving the performance of XML parsing in embedded systems. The various parsers available for minimum memory requirement as essential in real time embedded systems are discussed and compared in this paper. The experimental results show that VTD is the better solution as basic parsing approach. This approach can be extended in future for paring in real time embedded environment by combining it with existing strategies such as view management for approximated data, pattern tokenization, XML filtering schemes, etc. [22-24].

References

[1] Tak Cheung Lam, Jianxun Jaso Ding and Jyh - Charn Liu, "XML Document Parsing: Operational and Performance Characteristics", in IEEE Computer Society, 2008.

[2] Hazem M., El-Bakry and Nikos Mastorakis, "Performance Evaluation of XML Web Services for Real-Time Applications", in International Journal of Communications, Volume 3, Number 2, 2009.

[3] Esther Mınguez Collado, M. Angeles Cavia Soto, Jose A. Perez Garca, Ivan M.Delamer, and Jose L.Martınez Lastra, "Embedded XML DOM Parser: An Approach for XML Data Processing on Networked Embedded Systems with Real-Time," EURASIP Journal on Embedded Systems, Volume 2008.

[4] M. F. L Hufkens, "XML for embedded systems: the role of XML in distributed embedded networks", Master's thesis, Technische Universiteit Eindhoven, Department of Mathematics and Computing Science, August 2003.

[5] Wei Zhang, Robert A. Van Engelen, "An Adaptive XML Parser for Developing High-Performance Web Services" in Fourth IEEE International Conference on eScience , 2008.

[6] Zhang W. and R. van Engelen, "A Table-Driven Streaming XML Parsing Methodology for High-Performance Web Services", in IEEE International Conference on Web Services (ICWS'06), Sep 2006, pp. 197–204.

[7] Zhang W. and R. van Engelen, "High-Performance XML Parsing and Validation with Permutation Phrase Grammer Parsers", in IEEE International Conference on Web Services (ICWS'08), Sep 2008, pp. 286-294.

[8] Michael R Head and Madhusudhan Govindaraju, "Parallel Processing of Large-Scale XML–Based Application Documents on Multi-core Architectures with PiXiMaL", in IEEE Fourth International Conference on eScience, (eScience'08), 2008, pp. 261-268.

[9] Lelli F., Maron G. and Orlando S. , "Improving the performance of XML based technologies by caching and reusing information" in International Conference on Web Services, (ICWS '06), Chicago, Sep 2006, pp. 689-700.

[10] Gang WANG, Cheng XU, Ying LI, Ying CHEN, "Analyzing XML Parser Memory Characteristics: Experiments towards Improving Web Services Performance" in IEEE International Conference on Web Services (ICWS'06), 2006.

[11] Zhou Yanming and Qu Mingbin, "A Run-time Adaptive and Code-size Efficient XML Parser," in Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06), IEEE Computer Society, 2006.

[12] Zhou Dong, "Exploiting Structure Recurrence in XML Processing", in Proceedings of the Eighth IEEE International Conference on Web Engineering (ICWE'08), July 2008, pp. 311–324.

[13] Eric Jui-Lin Lu and Ying-Sheng Lee, "A Processing Rules Generator for an XML-based Mobile Database" in Proceedings of the IEEE International Conference on Networking, Sensing & Control Taipei, Taiwan, March 2004, pp. 106- 111.

[14] Giuseppe Psaila "Loosely Coupling Java Algorithms and XML Parsers: a Performance-Oriented Study" in Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06) IEEE Computer Society , 2006.

[15] T. Szalapski, S. Madria, and M. Linderman, "TinyPack XML: Real Time XML Compression for Wireless Sensor Networks", in Wireless Communications and Networking Conference (WCNC), IEEE, 2012, pp. 3165-3170.

[16] Fadi El-Hassan, and Dan Ionescu, "SCBXP: An Efficient CAM-Based XML Parsing Technique in Hardware Environments," IEEE Transaction on Parallel and Distributed Systems, Vol. 22, Nov 2011, pp.1879- 1887.

[17] Zhou Yanming and Qu Mingbin, "A Run-time Adaptive and Code-size Efficient XML Parser," in Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06), IEEE Computer Society, 2006.

[18] Zhou Dong, "Exploiting Structure Recurrence in XML Processing", in Proceedings of the Eighth IEEE International Conference on Web Engineering (ICWE'08), July 2008, pp. 311-324.

[19] Ma Jianliang, Shaobin Zhang, Tongsen Hu, Minghui Wu and Tianzhou Chen, "Parallel Speculative Dom-based XML Parser", 14th IEEE International Conference on High Performance Computing and Communications, 2012.

[20] Jie tang, Shaoshan Liu, Chen Liu, Zhimin Gu, and Jean-Luc Gaudiot, "Acceleration of XML Parsing through Prefetching", in IEEE Transactions on Computers, Volume 62, Number 8, Aug 2013.

[21] Vlado Altmann, Jan Skodzik, Peter Danielis, Nam Pham Van, Frank Golatowski, and Dirk Timmermann, "Real-Time Capable Hardware-based Parser for Efficient XML Interchange" in 9th International Symposium on Communication Systems, Networks & Digital Sign (CSNDSP), 2014, pp. 395-400.

[22] G. Suddul, N. Nissanke, and N. Mohamudally, "A pattern based tokenization model for XML parsing on mobile devices", in IEEE transactions, Sep 2013, pp. 1-5.

[23] Yu Xiaochuan, C. S. Alvin, "A time/space efficient XML filtering system for mobile environment", 12th IEEE International Conference on Mobile Data Management (MDM), Volume 1, 2011, pp. 184-193.

[24] Amol Bhagat and Bhaskar Harle, "Materialized view management in peer to peer environment," Proceedings of the ACM International Conference & Workshop on Emerging Trends in Technology, 2011, pp. 480-484.

[25] Akhil Rangan C K, Jayanthi J, "A Generic Parser to Parse and Reconfigure XML files", in IEEE conference on Recent Advances in Intelligent Computational Systems (RAICS), pp. 823-827, 2011.