Improving the Access to XML Contents of Large Electronic Publications

V. Fresno-Fernández, R. Martínez-Unanue, M. Paredes-Velasco,
J. Urquiza-Fuentes, and J. A. Velázquez-Iturbide

ESCET, Universidad Rey Juan Carlos
C/ Tulipan s/n, 28933 Móstoles -Madrid-
{v.fresno, r.martinez, m.paredes, j.urquiza, a.velazquez}@escet.urjc.es

Abstract. In this paper we present a technique to access the contents of large XML documents. This technique can be applied to domains where large amounts of data are handled, on condition that those contents only require to be updated in a scheduled or sporadic way. We have implemented it in a system, named ALEP (Access to XML contents of Large Electronic Publications). First, ALEP preprocess the XML document, then it is used together with DOM API. One of their advantages is that it can be integrated into any software for management XML electronic publications. We have compared ALEP with DOM and SAX API's improving both, time and space measures, with large documents of size greater than 18 MB megabytes acceding to a hundred of XML elements.

1 Introduction

Nowadays, XML [1] and its associated technologies are more and more used in Electronic Publishing. These technologies give favorable solutions to specific needs in many circumstances; however, they may also be inadequate in other situations. Thus, when we work with large XML documents, an internal representation of the contents and structure based on its complete storage in memory can be inefficient. In addition, if time of data access is a critical factor and many accesses to a large XML document are made, an API providing sequential access does not guarantee acceptable response times. A partial solution comes from technologies that focus on query languages with features developed by the database research community for semistructured data.

We have developed a general and efficient technique to access the contents of large XML documents that takes advantage of the exiting XML technologies. In our approach, we do not need a data structure storing the whole contents in memory, but a tree that represents the document internal structure. Every node of the tree only stores the information necessary to access directly those contents. On the one hand, the tree structure is obtained from an analysis of the XML document. On the other hand, the node information is determined from its location within the document contents. We have implemented it as a system capable of creating a direct access structure. This system, named ALEP (Access to XML contents of Large Electronic Publications), can be integrated into any software for management XML electronic publications, and it is used together with DOM API.

Next, main access methods to XML contents are briefly described.

1.1 Main Access Methods to XML Contents

DOM (Document Object Model) [2] describes a way to process, access and manipulate content of XML documents. Every XML document is a hierarchy of elements. DOM builds a tree representing the hierarchical structure of the XML document where elements and entities of the XML document are nodes of the tree.

The programmer who uses a DOM parser can access to the contents of the XML document as easy as accessing a tree structure. After parsing a XML document, a DOM parser returns the root node of the tree, which is the root element of the XML document. The DOM API [2] details how to manipulate a XML document in terms of nodes.

The work of a DOM parser consists on three steps: first it parses the document, ensuring the validity a correctness, then it stores the tree structure in memory and returns the root node, and finally, it returns everything the programmer asks using the DOM API. At the second step, the DOM parser stores the whole tree in memory, so accessing to the first node has the same cost as accessing the last node. There is a variant from DOM, called lazy DOM, which stores a node and its child nodes in memory only when the node is accessed, so in some cases it will be more efficient.

SAX [3] is a parser for XML documents based on capture and processing of events. Therefore, when we are working with a SAX parser, XML document information is a sequence of calls and events. There are two roles in this API: events producer and events consumer or events handler. The events producer is very simple: it is a call to a procedure of a class of some commercial library. The events consumer is written by the application programmer. When an event happens, this consumer will decide what to do. The producer will produce an event for every open tag, every close tag, every #PCDATA and DATA section. Besides, it also produces events for processing instructions, DTDs, comments, etc.

SAX consumes little memory, this is its main advantage. This memory expensive does not change between big or small documents. Another advantage is its data structure flexibility. DOM obligates you to use a tree structure to represent information contained in XML document, whereas SAX allows you to use your API to create data structure. However, it presents a disadvantage: SAX API programming is more complicated because you have to manage your own data structure.

XQL and XML-QL are both query languages of XML contents. **XQL** [4] was an early proposal for a simple query language designed specifically for XML. The basic constructs of XQL correspond directly to the basic structures of XML, and XQL is closely related to XPath, the common locator syntax used by XSL and XPointers. Since queries, transformation patterns, and links are all based on patterns in structures found in possible XML documents, a common model for the pattern language used in these three applications is possible. **XML-QL** ([5], [6]) is a query language for XML data to allow extraction, transformation and integration of XML data. It is designed to express queries to extract pieces of data from XML documents; it can express transformations to map XML data between DTDs and integrate XML data from different sources. The language has some constructs similar to SQL, and borrows features of query languages recently developed by the database research community for semistructured data.

1.2 Use Points in Large Publications

DOM uses a lot of memory, spends few time and is easy to use. Using a DOM parser with big documents will result on a great amount of memory used, because DOM stores the whole tree in memory; so in terms of memory usage DOM is unefficient. In terms of time going from one node to another, DOM is very fast because the tree is stored in memory, and going to a node is the same as read other memory address. In terms of effort spent on learning DOM API, DOM is easy because DOM builds a tree structure, which is a data structure known by every programmer, and very easy to use.

When we are working with big documents, SAX presents some deficiencies or disadvantages. In first place, SAX parser does not provide random access to XML data. If you want to go from a node to another one, you always must start from the root node. In large XML documents, this characteristic causes that the application is very slow because you must repeat parser process every time. An other disadvantages is that you have to provide tasks and actions in order to manage the elements order in your XML document.

Finally, the query languages designed specifically for XML are appropriate for permitting open queries about the contents of XML documents, which can be expressed by means of patterns or constructs of such query languages. However, some domains and applications could not need such potentiality, owing to all possible contents access are planned in some way.

The organization of the paper is as follows: Section 2 introduces our access method ALEP; in Section 3 we present the experimental results; finally, Section 4 summarizes the conclusions drawn from the work carried out.

2 The Access Method ALEP

2.1 Motivation

We have involved in a project supported by the Spanish Research Agency which have as one of its main goals the development of an electronic book about computer programming. We thought that the functionalities and facilities we had designed could be solved with XML and appropriate technologies. Such e-book should be able to offer at least a sophisticated navigation system, based on different techniques: navigation controls, search facilities, user-defined bookmarks, ... These facilities require an efficient access to XML contents of the e-book. The efficiency we are talking about is referred to both, time and space. On the one hand, navigation amongst contents must be as fast as possible. On the other hand, the necessary RAM memory to run the e-book application must not be greater than the standard memory of a typical PC existing in any house or in any practice room.

Once we fixed our requirements, we identified the contents characteristics of this e-book in order to take advantages of the later. From this point of view, the main contents characteristic is its potential static nature. The chapter, sections, examples, exercises, that is, the real contents, will be always the same; although different users can be have different views of those contents. Different users can customize them by means of different classes of annotations. Nevertheless, customisations do not affect what we have called real contents because they can be registered separately.

The potential static nature of the contents, permits improve the access efficiency by means of the use of information about the localization of those contents. In next subsections our approach and the system that implemented it are detailed. This system can be applied to domains where large amounts of data are handled. These data need not be static, but require to be updated in a scheduled or sporadic way. Some others examples of these domains are: e-books, technical manuals, administrative and legal documentation, warehouses, or exercises collections.

2.2 ALEP Description

We propose a general and efficient technique, called ALEP, which this main is to overcome the limitations that DOM and SAX presents when large XML contents are handled and many accesses are needed.

ALEP tries to make use of advantages of the method DOM in time reducing its limitations in memory. This method works in two phases:

1. In a preprocessing, the Direct Access Document (DAD) associated to the XML document is created. DAD is another document with the same hierarchical structure as the original document, where #PCDATA contents are exchanged by the information needed to access them in the original document. The rest of the original document does not change. This information is the position in the original document of the first character of the content, and the number of characters until the end of the content. In this way, a new document is generated with the same structure that the original, but in this case some of their elements are pointers to the contents of the original XML document.
2. Once DAD has been generated, as it is a XML document, it can be processed with DOM or SAX. We use DOM because is very easy and powerful. DOM stores all the DAD document structure and content in memory; however, the content now is the minimum necessary to find the original XML content, so the memory occupied is the minimum necessary to process the document as it was the original one. A method is added to the XML application in order to recover the real content from the original XML document using the DAD pointers.

One of the advantages of DAD is that the programmer have not to learn a new method to process XML documents, because he can use DOM to access DAD nodes. When the programmer asks the #PCDATA content of an element our system do: (1) get the beginning and the length of the content, (2) get this part of the original document and return it to the programmer.

We have developed ALEP using Java as the programming language and Xerces [7] as the XML parser.

3 Experimental Results

3.1 Evaluation metrics

An XML contents handle system must be evaluated in terms of two fundamental factors: the time employed in the access and the memory needed. We introduce a effectiveness evaluation function in terms of those resources. A relative evaluation between two systems could be made in order to determine whether a specific system can be more effective than other one. An objectivist comparison need a range where to establish such comparison. We propose an effectiveness evaluation function enclosed in the interval $(0, 1)$, which take the following expression:

$$f(t, Ram) = 1 - \exp^{\frac{k}{t \times Ram}}$$

where k is a constant that depends on the units of the variables. As an example, when time is measured in seconds and memory in MegaBytes, the constant k must

take a value 100. In this way, if a system is evaluated with a value near to 1 then can be considered like an efficient system, where exists a good ratio between the time and the memory employed. The function what we introduce here can be used to evaluate any computer system where a inverse relation in time and memory are wanted.

Basing on this evaluation function, an empirical study is carried out in order to compare the behaviors of the DOM, SAX and ALEP systems to access to the contents of the XML document.

3.2 Tests Description

The test has been developed in a processor Pentium III workstation with 128 MG Ram memory. The operating system used has been SuSE Linux 7.3. The XML documents we have work with are documents validated with the DocBooK DTD. The proof set was generated from a XML document corresponding to a complete chapter of an e-book for teaching computer programming.

The experiments have included different documents size. The documents size varies from 81 KB to 20 MB. The number of elements accessed in each test is 100. We have chosen this value because in the e-book applications domain is usual access to more than one page; so the access to a group of 10 pages (with paragraphs, images, code, references, sections, examples, ...) we have calculate that can be need the access to 100 elements approximately.

We have compared the behavior of the DOM, SAX and ALEP systems with respect to time and memory. In addition we use a function to evaluate both time and memory.

3.3 Results

The results of the experiments with respect to the evaluation function, time and memory can be seen in Figure 1, Figure 2, and Figure 3 respectively.

With documents equal or bigger than 12 MB, DOM API produce a memory lack in this computer. Lazy DOM produce similar memory lack with documents equal or bigger than 18 MB. ALEP is capable of processing larger XML documents than DOM and Lazy DOM are. SAX is also capable to manage large documents, however it needs so much time that it could be inoperative in several applications.

Figure 2 relates memory needed with respect to document size. It can be seen that except SAX, which has an almost constant behavior, DOM and Lazy DOM increase memory requirement faster than ALEP.

With respect to the time, Figure 3 shows the experimental results. Clearly SAX presents the worst behavior; DOM and Lazy DOM require less time that ALEP, however the growth is very similar in these three cases.

Lazy DOM shows the best behavior relating time and memory (see Figure 1) when the memory computer is enough to process the document. ALEP is the one which manages largest documents in a feasible time in a computational point of view.

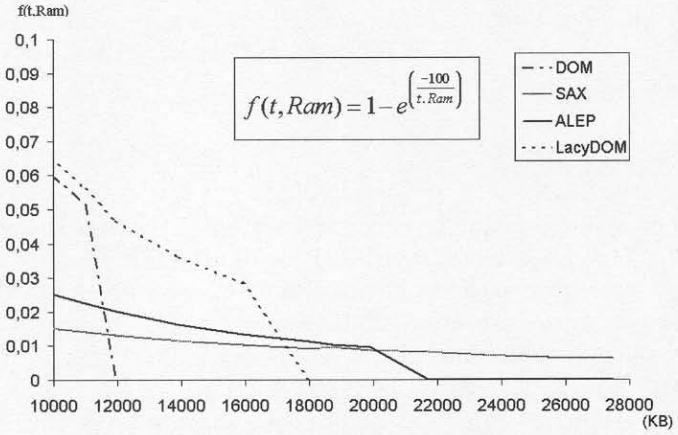

Fig. 1. Behavior respect to the time and memory.

4 Conclusions and Future Work

We have carried out and evaluation of the two main APISs (DOM and SAX), and our system ALEP with the contents of an e-book about computer programming with the DocBooK DTD. ALEP preprocess the XML document for creating a Direct Access Document which is used with DOM API.

The experiments have included different XML documents size. The results show that ALEP process larger documents than DOM and Lazy DOM, and ALEP does it in less time than SAX does.

ALEP can be applied to any domain where large amounts of data are handled. Because of the ALEP preprocess, the main requirement is that the contents or data need to be updated in a scheduled or sporadic way. Some examples of these domains are: e-books, technical manuals, administrative and legal documentation, warehouses, or exercises collections.

With regard to the future, we will focus on reducing the memory requirement of ALEP without increase time, in order to improve efficiency in the e-book domain. We will carry out more experiments taking into account the ratio between number of tags and its contents.

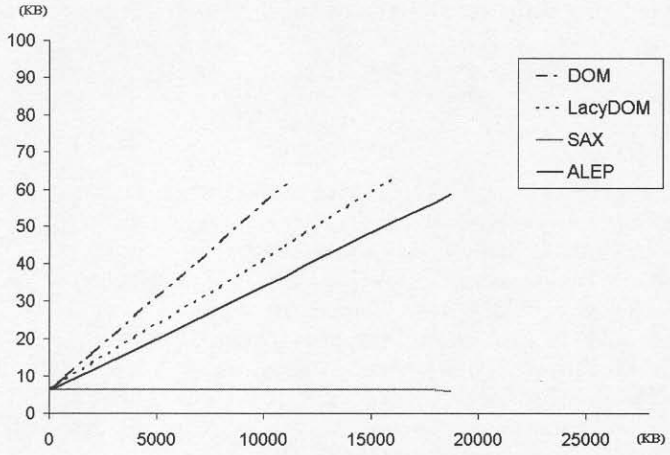

Fig. 2. Behavior respect to the memory.

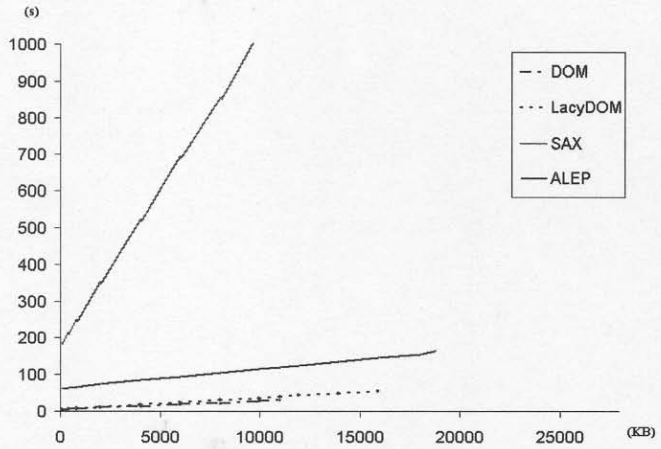

Fig. 3. Behavior respect to the time.

Acknowledgements

This research is being supported by the Spanish Research Agency, project LIBRE, TIC-2000-1413.

References

1. *Extensible Markup Language (XML).* `http://www.w3.org/XML/`, 1998.
2. *Document Object Model (DOM).* `http://www.w3c.org/DOM/`.
3. D. Brownell. *SAX2.* O'Reilly & Associates, 2002.
4. E. Derksen, P. Fankhauser, E. Howland, G. Huck, I. Macherius, M. Murata, M. Resnick, H. Schning. "XQL (XML Query Language)". `http://www.ibiblio.org/xql/xql-proposal.html`, 1999.
5. A. Deutsch, M. Fernndez, D. Florescu, A. Levy, D. Suciu. "XML-QL: A Query Language for XML". `http://www.w3.org/TR/NOTE-xml-ql/`, 1998.
6. A. Deutsch, M. Fernndez, D. Florescu, A. Levy, D. Maier, D. Suciu. "Querying XML Data". *IEEE DATA Engineering Bulletin*, 22(3), 1999.
7. *Xerces Java Parser.* `http://xml.apache.org/xerces-j/index.html`.