Engineering Succinct DOM

O'Neil Delpratt

(Joint work with Rajeev Raman, Naila Rahman)

Department of Computer Science, University of Leicester

Introduction

XML is a standard format for data exchange and storage. XML documents are processed by a number of applications in the following manner: the XML document is parsed, and a tree representation of the XML document is created within the memory of the computer. This representation is then accessed through the standard DOM interface.

The DOM interface is very flexible, and is commonly used for XML processing. Our focus is on *static* XML documents --- while DOM does have functionality that allows (fairly arbitrary) changes to the XML document, this functionality is not frequently used. Indeed, there are a few DOM implementations for static documents.

We discuss the advantages and disadvantages of existing implementations of the DOM and describe a new approach in our DOM implementation that is based upon *succinct* data structures.

Motivations

A major disadvantage of most implementations of the DOM is a high memory requirement, referred to as ``XML bloat''. The in-memory DOM representation of an XML document can be many times larger than the XML file itself, for example figure 1. This means that even moderately large XML documents cannot be processed within the main memory of a reasonably high-end machine.

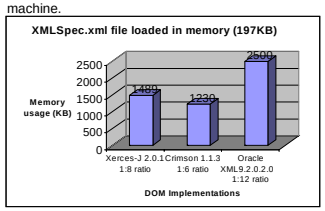

Figure 1. Test of Memory usage for standard DOM implementations. Loaded the XML specification in a DOM object in java [1].

Given the DOM tree of the simple XML document in figure 2, many implementations of the DOM use a pointer-based representation for associations between the various data elements.

```
<book catalogue="XML">
 <author>
Delpratt&amp;&cop;
 </author>
 <title> SDOM Design
<title>
 <year> 2007 </year>
</book>
```

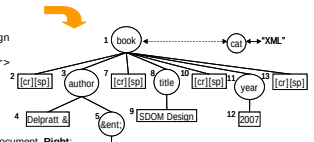

Figure 2. Left: Simple XML document. **Right**: DOM tree structure of Left, with numbering. cr: carriage return char, sp: space char

These pointers are a primary cause for the ``XML bloat''. A single node in the tree representation of an XML document may have pointers to its parent, first-child, and its next and previous siblings, among others. Each pointer would require 32 or 64 bits, so one would expect more than 20 bytes or 40 bytes per node.

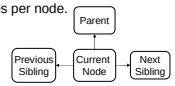

Figure 3. Pointer based representation of the DOM node.

Succinctness

We describe a implementation of the DOM that does not require the use of node-pointers, and is based upon *succinct* data structures.

Succinct data structures use the information-theoretically minimum number of bits to encode a object. For example, an *ordinal* tree on n nodes is a rooted tree, where the children of a node are ordered from left-to-right (XML documents are essentially ordinal trees). The lower bound for representing an ordinal tree on n nodes is $2n$ - O(log n) bits [1,5].

This is much better than the pointer-based representations (described earlier), which would use asymptotically $4n \log_2 n$ bits.

Succinct Tree representation

Parentheses Tree

Construction:

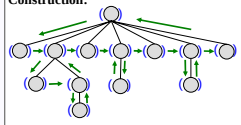

Figure 4. Parenthesis tree representation[2] of XML doc (figure 2)

Depth first Order sequence of tree:

2n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
n	1	2		3	4		5	6				7		8	9			10		11	12			13		
	(	(	)	(	(	)	(	(	)	)	)	(	)	(	(	)	)	(	)	(	(	)	)	(	(	)

We consider the parenthesis sequence as a sequence of bits(called bit-string)

0010010011101001101011

Facts:

- Nodes represented by the opening parenthesis.
- Node location in the tree are known by their position in the parenthesis sequence (p) and by their count of opening parenthesis (d). `Pair{d,p}`

Parentheses tree operations

A parentheses representation supports operations for a parenthesis location: ENCLOSE, FINDCLOSE, FINDNOPEN and INSPECT.

Providing us with tree navigation:

- Parent, first-child, next-sibling, previous-sibling. Iterator: next-node, previous-node

Example:

ENCLOSE(i): Returns the position of the opening parenthesis of the pair that tightly encloses the parenthesis i.

DOM Node operation getParentNode:

Nodes represented by 2 numbers: pair(d,p)

```
getParentNode(d,p){
  p'= ENCLOSE(p)
  d'= d-(p-p'+1)/2
  return pair(d',p')
}
```

Get parent node of 7th node at position 12:

getParentNode (<7,12>): p':=ENCLOSE(12)=1; d':=7-(12-1+1)/2=1; return (1,1);

SDOM Components

We have implemented Succinct DOM (SDOM) in C++. The process of building the SDOM data structure from an XML document is via a SAX parser. Figure 5 shows the architecture of SDOM. We have already discussed the tree structure (STree), we now give an overview of the other components.

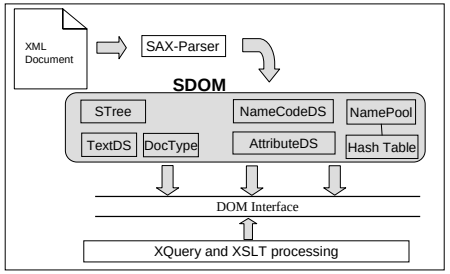

Figure 5. Overview of parsing XML document into SDOM components.

TextDS: The DOM text nodes consumes a huge part of memory used for representing XML documents (~30-60% of tree).

Standard techniques exist where we may compress the textual data concatenated (FM-Index [6]), however providing pointers to where each text data start and end is very costly. We use succinct data structures [3] in place of these explicit pointers to the lists of attributes values and text nodes.

NameCodeDS, NamePool, HashTable: We use a similar data structure of tinyTree Namepool (http://saxon.sourceforge.net/). We make use the hash-tables for XML names, by mapping them to 32-bit namecodes and we provide optimisations for these namecode storing them compactly via shorter codes with-in integer arrays.

AttributeDS: Attributes themselves are not apart of the DOM tree, but are referenced to the elements where they are defined through a NameNodeMap. We provide a mapping of attributes to the elements they belong to using a tree-like bit-string, illustrated in Figure 6.

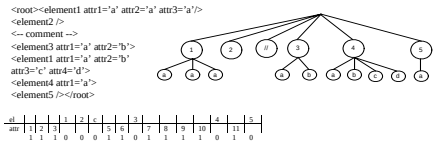

Fig. 6. Top Left: Example XML doc with elements and attributes. Top Right: Tree-like mapping of elements attributes. Bottom: Bit-string of the AttributeDS representation.

XML docs	Original file size	SDOM Components	Text (uncompressed)	SDOM Total	Xerces C++
Mondial-3.0.xml	1081KB	160KB	688KB	848KB	4239KB
Partsupp.xml	2253KB	135KB	1088KB	1223KB	4518KB
Orders.xml	5243KB	389KB	1488KB	1877KB	12766KB

Figure 7. Test XML files and their sizes. SDOM space usage with no text compression applied. SDOM memory usage compared to Xerces-C.

Experiments

Experiments on our version SDOM was done by reading and storing an XML document in main memory, and traversing its tree. Reading element's attributes were also accessed

XML docs	SDOM	Xerces	SDOM Slowdown relative to Xerces-C
Mondial-3.0.xml	0.016	0.005	2.94
Partsupp.xml	0.036	0.007	4.48
Orders.xml	0.111	0.023	4.80

Figure 7. Test files. Preliminary tree traversal performance times (milliseconds). SDOM slowdown wrt. to Xerces-C.

Conclusions

Motivated by succinct representations we have discovered new uses for the application of XML document content and its structure. By the engineering of SDOM.

SDOM provides flexibility in XML processing by having tuning parameters in the SDOM components. This greatly reduces space usage where memory is limited or increases space usage where performance is critical.

We have shown that Succinct trees improve the space complexity without compromising too much on query time. Also, to represent XML trees close to optimum space, while supporting a wide range of operations efficiently.

Literature cited

[1] Dennis M. Sosnoski, http://www.sosnoski.com/opensrc/xmlbench/index.html, 2006.

[2] O'Neil Delpratt, Naila Rahman, Rajeev Raman, Engineering the LOUDS succinct tree representation. In proc. *WEA 2006*, LNCS 4007, p 134-145, Springer, 2006.

[3] O'Neil Delpratt, Naila Rahman, Rajeev Raman, Compressed Prefix Sums. *SOFSEM 2007*, LNCS 4362, p 235-247, Springer.

[4] Jacobson, Space-efficient static trees and graphs. In *Proc. 30th FOCS*, 549-554, 1989.

[5] R. F. Geary, N. Rahman, R. Raman and V. Raman. A simple optimal representation for balanced parentheses. In *Proc. 15th CPM*, LNCS 3109, pp. 159-172.

[6] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, S. Muthukrishnan, Compressing and Searching XML Data Via Two Zips, In: Proceedings of the 15th International World Wide Web Conference, ACM Press, Edinburgh, UK, May 2006, ISBN 1-59593-323

Acknowledgments

I thank GOD for the understanding He has given.

I would like to thank Professor Rajeev Raman as my supervisor for the advice and support. Both Rajeev Raman and Naila Raman's collaborative work has made the engineering of SDOM possible.

For further information

Please contact *ond1@mcs.le.ac.uk*. More information on this and related projects including published paper can be obtained at *www.cs.le.ac.uk/people/ond1*

Poster version in PDF-version www.cs.le.ac.uk/people/ond1/XMLcomp/XMLPrague_SDOM_Poster.pdf

University of Leicester

Developed: 12th June 2007