Succinct DOM 1.0 library

O'Neil Delpratt
(Joint work with Rajeev Raman, Naila Rahman)

Department of Computer Science, University of Leicester

Introduction

XML is a multipurpose data format that is well-suited to the representation of complex, hierarchically structured data. Its uses in data exchange, storage and retrieval have reached much further than its creators may have anticipated.

The Document Object Model (DOM) is a W3C specification for processing XML documents. The DOM is a tree-based representation of the XML document loaded in memory, and has a rich set of APIs to navigate the tree document.

We address the shortcomings of existing DOM implementations, such as excessive memory usage and the dynamic costs of node object creation, whilst providing fast navigational and data access support. Our focus is on a space-efficient tree-based representation of XML documents using *succinct* data structures to efficiently support the DOM read-only operations.

Motivations

XML documents are often much larger than equivalent binary formats. This 'XML bloat' becomes a problem when communicating XML data or when using XML in mobile devices that have limited memory space.

DOM maintains the tree representation of the document in main memory, as it is fast. However, existing implementations suffer from a high memory usage: For example, the memory usage of the standard Xerces-C implementation [5] is up to 10 times the size of the XML file. Such a big increase is problematic when processing XML documents using DOM, even on a desktop PC or server.

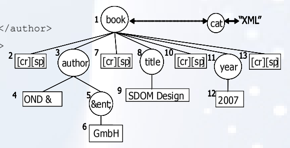
Succinctness

Our approach to representing XML documents with low memory footprint relies upon *succinct data structures*, which use the information-theoretically minimum number of bits to encode an object. The structure of an XML tree can obviously be represented using 2 bits per tree node [2] (See Figure 1). Current DOM implementations use a pointer-based approach: each node has pointers to its parent, first-child, next-sibling, previous-sibling etc. This uses memory (32 or 64 bits per pointer) but is fast (navigation is just a pointer dereferencing).

Based on several years' research, we have implemented *succinct trees* which use < 3 bits per node, but support navigation almost as fast as a pointer dereferencing (however, updates are not yet supported).

Using succinct trees and other "succinct" building blocks we are able to represent nodes in <14 bits per node, rather than 224.

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



f(*i*)	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
i	1	2		3	4		5	6				7		8	9			10		11	12			13		
Parentheses	(	(	)	(	(	)	(	(	)	)	)	(	)	(	(	)	)	(	)	(	(	)	)	(	)	)
BitString	0	0	1	0	0	1	0	0	1	1	1	0	1	0	0	1	1	0	1	0	0	1	1	0	1	1

Figure 1. Sample XML document. DOM tree structure. Succinct tree representation and equivalent bit-string. [cr]:carriage return. [sp]: space character.

Succinct DOM (SDOM) Library 1.0

SDOM supports the read-only operations of an almost complete DOM level 3; which maintains interfaces for both the *Core* and *XML* modules of the DOM APIs. In addition, we support the *TreeWalker* API of the extended DOM Traversal module.

SDOM libraries are available for downloaded at the webpage:

https://lra.le.ac.uk/handle/2381/3363

The space-efficient succinct data structures written in C++ are highly optimised. In summary, SDOM consists of a:

- ✓ Succinct tree (Stree) representation
- ✓ Namecode data structure, adapted from Saxon [4]
- ✓ Attribute to element node mapping data structure
- ✓ Textual data structures. Fast access to individual text. Facility to compress text in blocks using Bzip2; this we call SDOM-CT.

Within the DOM interface SDOM-(CT) has intermediate interfaces, which call directly the succinct data structures. These interfaces are similar to the *NodeInfo* and *DocumentInfo* interfaces used in Saxon to directly access the TinyTree data structure.

We aim into a high-level application, such as a plug-in to Saxon.

Features of SDOM

✓ SDOM can be used as a standard-alone DOM.

However navigation between nodes requires the creation of node objects, which must be maintained. To avoid this problem we recommend the use of the *TreeWalker* class for navigation; this has an iterator like behaviour, so node objects are not created by navigation operations.

We show in the following code snippets two versions of the operation getFirstChild() from the *SDOM-Node* and *SDOM-TreeWalker classes*. We observe that SDOM-Node requires the creation of two objects when returning an SDOM node object, whereas SDOM-TreeWalker simply fetches the node number from the succinct tree (STree), then assigns the current node to this value, if the node exists.

✓ The STree representation of SDOM efficiently supports the XPATH axes (excluding namespace) – however interface development required.

First-Child operation in SDOM-Node class:

```
SDOM_Node * SDOM_Node::getFirstChild()
{
  nodej = STree.firstChild(nodei);
  if(nodej.node_n == 0)
     return NULL;
  SNodeImpl * node = new SNodeImpl(STree,nodej);
  return (new SDOM_Node(node));
}
```

FirstChild operation in SDOM-TreeWalker class:

```
SSDOM_Node * TreeWalker::getFirstChild()
{
  nodei = STree->firstChild(currentNode);
  if(nodei.node_n == 0)
     return NULL;
  currentNode = nodei;
  currentNodeImpl->setNodeNr(currentNode);
  return currentNodeImpl;
}
```

Experimental Evaluations

A full scale experimental evaluation of SDOM-(CT) has been provided in [1].

Space Usage

In Table 1, we show the space usage of SDOM-(CT) compared to Xerces-C, Saxon's TinyTree and a reputable XML compressor, called XMILL [3]. We show space usage for a document-centric file (*Orders.xml*) and a data-centric file (*Treebank_e.xml*).

Table 1. Space usage of XML representations. % of memory usage w.r.t to file size.

File	Size	SDOM	Xerces-C	Saxon	SDOM-CT	XMILL
Orders.xml	5MB	37%	451%	157%	17%	12%
Treebank_e.xml	82MB	84%	866%	260%	54%	30%

Observations:

➢ Xerces-C is much larger than the XML file. Saxon is considerably better than Xerces-C.

➢ Memory usage of SDOM is considerably less than the file size. SDOM-CT provides further gains.

➢ SDOM-CT is *query-friendly* (i.e. supports queries without requiring decompression) and is comparable to XMILL, which is not query-friendly.

Running times

In Figure 2, we shows the results of a document-order traversal of the XML documents, which includes a simple substring test. Results are for the NextNode operation of SDOM-TreeWalker and the navigation operations of SDOM-Node.

Observations:

➢ SDOM is always within a factor of 2 of Xerces-C.

➢ Using the NextNode operation of SDOM-TreeWalker, we are closer to Xerces-C. [1] shows SDOM is better for larger files.

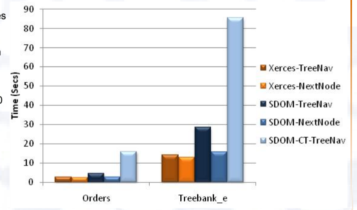


Figure 2. Running times for DOM full test, which includes search for a uncommon substring in all text nodes.

Traversals were run 50 times for Orders and 10 times for Treebank_e

Conclusions

SDOM is a fast in-memory representation of XML documents with small memory footprint. The current implementation is close to being a plug-in replacement for a standard DOM implementation, in applications that requires read-only operations , with very little penalty in terms of CPU usage.

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.

Current development is to optimise the speed and memory usage of the pre-processing of SDOM. Finally, in addition to the tests, we are working towards a plug-in to an XSLT processor, such as Saxon.

Literature cited

[1] Delpratt, O., Raman, R., and Rahman, N. 2008. Engineering succinct DOM. In *Proc. of the 11th International Conference on Extending Database Technology: Advances in Database Technology*. EDBT 2008. ACM Press.

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

[3] Liefke, H. and Suciu, D. 2000. XMill: an efficient compressor for XML data. 2000. In *Proc. of the ACM SIGMOD international Conference on Management of Data* SIGMOD 2000. ACM Press.

[4] Saxon. http://saxon.source.forge.net/

[5] Xerces C++ Parser. http://xerces.apache.org/xerces-c/

For further information

Please contact *ond1@mcs.le.ac.uk*.
Poster version in PDF-version
www.cs.le.ac.uk/people/ond1/XML.comp/XMLPrague09_SDOMPoster.pdf