Large XML on Small Devices: Techniques Developed in the Fuego Core Project

Tancred Lindholm

May 28, 2007

Storage capacity on limited and mobile devices has risen rapidly in recent years. The amount of available processing capacity is, however, still low, as it is limited by battery capacity. In this talk we look at techniques for efficiently processing the increasing amounts of data in XML format that can be fit on a mobile phone. The techniques were developed by the author and Jaakko Kangasharju as part of the Fuego XML stack in context of the Fuego Core project on mobility middleware.

The XML processing techniques included in the Fuego XML stack that we present are: processing sequences of XML particles, parser/serializer byte stream access, parsing in a random access manner, using delayed tree structures and building mutability on these, as well as document packaging for synchronization.

In many cases, it is unnecessary to construct the tree structure corresponding to the XML document in order to perform a processing task. Instead, the document may be processed as a sequence of XML particles, such as document start, element start, element end, content, and so on. In the Fuego XML stack, the XAS XML parsing and serialization API supports such sequential processing. We have developed an XML efficient differencing application based on sequential XML processing [1].

One way of embedding binary data in XML documents is to use Base64-encoding, and embed the resulting text as a text node in the XML document. However, this large text object frequently becomes a bottleneck in XML parsing and serialization. This situation can be remedied by providing access to the raw input and output streams of the XML parser and serializer. Care has to be taken to preserve a valid XML context when switching between byte- and XML-level processing.

In some cases, XML documents are too large to be accessed in a sequential manner, in which case a parser that can be quickly positioned at

specific locations in the document is advantageous. This technique is especially useful in combination with delayed in-memory structures for the XML data, as one may then implement on-demand and selective loading of XML data.

The Fuego XML stack includes a type of delayed tree structure, called the *reftree*, and associated APIs for manipulating reftrees. This functionality allows easy development of applications that load tree structured data on demand and need to update the trees in a memory-efficient manner. The latter is accomplished with the *tree change buffer* structure. Furthermore, reftrees become very efficient tools when combined with the byte access and random access parsing capabilities of the lower layers.

The fundamental idea of the reftree is simple: to use a special kind of *reference node* to refer to nodes and subtrees from another tree structure. There are two kinds of references: *node* and *tree* references, where the former refers to a single node and the latter to an entire subtree. The API provides a set of primitive operations for manipulating these references, e.g., to combine several reftrees into one, or to reverse the role of the referencing and the referenced tree. Reference nodes do not automatically expand themselves, to shield the application developer form potentially costly expansion going on behind his back.

On the topmost layer we have the Random Access XML Store component, which provides packaging of XML documents with associated satellite files (such as images), and document-level operations, such as versioning, version recall, and synchronization. The packaging mechanism allows for easy migration of compound documents.

As a demonstration of this, we have written an XML editor that runs on Nokia 9500 mobile phones and has been used to edit XML files of up to 1GB on this platform. The editor uses on-demand loading of the document tree when viewing, and wraps a change buffer on top of the document tree for editing. The in-memory tree resulting from edits uses reference nodes to refer to subtrees and nodes that remain unchanged from the original tree, thus remaining compact despite the large size of the original tree.

References

[1] Tancred Lindholm, Jaakko Kangasharju, and Sasu Tarkoma. Fast and simple XML tree differencing by sequence alignment. In David F. Brailsford, editor, *ACM Symposium on Document Engineering*, pages 75–84, October 2006.