Introducing a Java-based Simple API for Binary REpresentations (SABRE)

Michael Hartle, Max Mühlhäuser
Fachgebiet Telekooperation
Technische Universität Darmstadt
Hochschulstr. 10
D-64289 Darmstadt, Germany
mhartle,max@tk.informatik.tu-darmstadt.de

Abstract

This paper presents the Simple API for Binary REpresentations (SABRE) in its initial version 1.0. SABRE defines observer-pattern interfaces for processing hierarchically structured, binary-oriented documents, comparable to the Simple API for XML (SAX). By modularizing processing steps into seperate stages of a streaming pipeline, SABRE facilitates parsing, transforming and serializing of large binary documents.

Its applicability for multimedia documents is demonstrated through its use case, the generation of Apple QuickTime multimedia documents, and an example for a selected portion of such a document is given.

1. Introduction

Current binary document formats such as for office or multimedia applications are complex and thus typically nontrivial to parse to and serialize from a memory representation. Implementing and maintaining conforming software is usually labour-intensive and error-prone. Furthermore, the size of binary documents can easily exceed the amount of main memory available in a standard PC system, especially for some applications in the multimedia field.

A solution to these problems lies in the modularization of either parsing or serialization process into seperate stages that are connected in a streaming pipeline. Stages for the parsing process gradually make the implicit structure in the binary document explicit which can then be used for further processing, whereas stages for serialization work in reverse.

By implementing components for each processing stage, functionality is encapsulated and its complexity hidden from other stages. These components can then be implemented, tested and maintained seperately. Instead of using model-based representations that simplify processing but do not scale well regarding memory requirements, event-based approaches can be used that pass only small portions of the overall representation at a time, therefore usually requiring more sophisticated processing.

Such a solution raises the question of appropriate interfaces to standardize and facilitate the stream-oriented processing of hierarchically structured binary documents in a component pipeline. Comparing binary-oriented to character-oriented processing, the latter has seen the standardization of concepts and interfaces related to the *Extensible Markup Language* [4] (XML) in recent years, whereas no comparable process has taken place for binary-oriented processing.

This paper first motivates the need for such an application programming interface (API) and refers to related work. Then the SABRE interfaces are specified in Java and a use case on the generation of Apple QuickTime movies with examples on SABRE is provided. The paper closes with conclusions.

2. Related Work

XML-centric technologies such as *DOM* [2], *SAX* [1], *XmlPull* [9] and *StAX* [5] offer standardized interfaces for accessing and updating hierarchically structured, character-based XML documents. This paper categorizes their concepts into model-/event-based, free/fixed traversal and push-/pull-based as shown in Figure 1 (compare [3, 11]):

- model-/event-based: Model representations (DOM) normally hold the complete document in memory, thus limiting input size. Event-based approaches (SAX, XmlPull, StAX) pass single document elements at a time and thus scale better. Processing can start directly with first event, yet non-trivial processing often requires data to be held explicitly between events until processing can be completed.

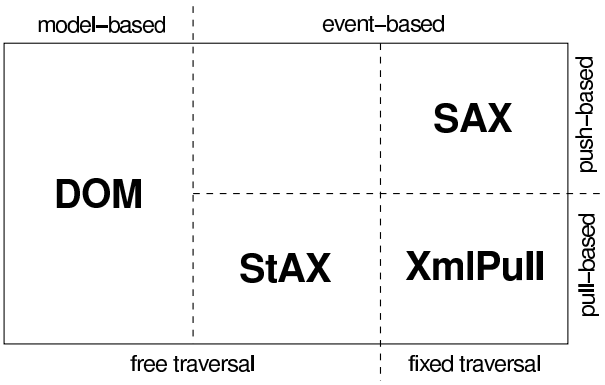

Figure 1. Categorization of processing APIs

- free/fixed traversal: Free traversal approaches (DOM, StAX) allow a processing algorithm to control traversal and traverse only the subset of the document which is relevant for processing. In contrast, fixed traversal approaches (SAX, XmlPull) require a processing algorithm to be adapted for the traveral order and direction, and to traverse the document including possibly irrelevant document elements.
- push-/pull-based: In push-based event approaches (SAX), the document-centric event source is responsible for pushing events for the entire document. Pull-based event approaches (StAX, XmlPull) reverse the flow direction, with the task-centric event sink pulling events only as and if needed.

The processing solution outlined in the introduction and presented in the following section under the name of SABRE, can be categorized as an event-/push-based, fixed traversal, binary-oriented processing API which is analogous to SAX.

3. Architecture

3.1. Interfaces

SABRE specifies a set of interfaces and a single exception class as shown in Figure 2 for software components to implement and use. Components receive events on structure and content during the fixed and complete traversal of a binary document by implementing the *StreamHandler* interface, which is an aggregation of the *StructureHandler* and *ContentHandler* interfaces, respectively.

StructureHandler offers notification methods for starting and stopping the document itself as well as hierarchical elements. In the current version of this API, the *Element* interface parameter passed when starting an element just has to

Figure 2. UML model of SABRE 1.0

provide an object as identification. Elements for themselves are explicit markup of data that is passed inbetween element start and stop.

ContentHandler offers notification methods for passing data as well as placeholders and a method for returning the current position in the bytestream. Data is passed encapsulated as a reference implementing the *DataReference* interface which offers methods for retrieving the data length and create an input stream to access the data. Data references enable the lazy evaluation of potentially large or remote content. When generating binary documents, it can be convenient not to pass data directly, but to pass a placeholder first and later provide the actual data, e.g. for format determinants. The *Fixup* returned when passing a placeholder can be used later in order to pass the actual data and close the fixup when complete.

In case of errors during processing, all interface methods may throw an *HandlerException*. Components are to be designed to pass on events to a successor as seen fit in order to work in a processing pipeline.

3.2. Usage

Central to the seperation of processing into seperate stages is the annotation of binary data and annotation-dependend processing via the *StructureHandler* interface in each stage as necessary.

For serialization pipelines, fully annotated binary data is entered into the pipeline and processed step by step, where annotations are modified as required for further process-

Figure 3. Pipeline of SABRE StreamHandler components used for the generation of Apple QuickTime movies

ing or removed. Likewise, parsing pipelines would need to parse the binary data and enhance its implicit structure with explicit annotations to trigger processing at a later stage. Mixed designs involving parsing, transformation and serialization are also possible.

3.3. Utilities

During the first use of the presented architecture, several reusable implementations were developed to simplify working with SABRE. This includes a base class for the development of chainable components (*ChainingStreamHandler*), ready components for debugging (*DebugStreamHandler*, extraction of data in elements (*CopyingStreamHandler*) and extraction of structure (*XMLAtomHandler*). Other classes mainly are DataReference implementations for primitives (e.g. *WordDataReference*), for files or portions thereof (*FileDataReference*) or remote content that can be referenced by an URL (*URLDataReference*).

4. Application

4.1. Overview

The need for SABRE arose in the context of the *Digital Lecture Hall* [10], an e-learning system for lecturers and students in presence lectures with extensive support for history-enabled presentations and pen-based annotations, electronic interaction and the recording of the presentation. SABRE is used for the serialization of media data to recordings in the QuickTime format [8] which provides for the synchronisation of MPEG-4 Visual video, MPEG-4 AAC audio and PNG slides. The result is a QuickTime movie that can be played back in the free Apple QuickTime Player.

4.2. Example

In the QuickTime file format, so-called *atoms* form the building blocks for the document. An atom consists of a 4 byte unsigned value representing the overall atom length in bytes, followed by a 4 byte type and an type-specific body, possibly containing fields and child atoms.

```
// Start edit list atoms
streamHandler.startElement(QTConstants.EDTSAtom));
streamHandler.startElement(QTConstants.ELSTAtom));

// Pass the ELST atom flags field
streamHandler.data(new WordDataReference(0));

// Pass a placeholder for the edit count determinant
fixup = streamHandler.fixup(new WordDataReference(0));

...

// For a single edit, pass fields for duration,
// starting time and playback rate
streamHandler.data(new WordDataReference(duration));
streamHandler.data(new WordDataReference(startingTime));
streamHandler.data(new WordDataReference(playbackRate));
editsCount++;

...

// After all edits have been passed, close the fixup
fixup.data(new WordDataReference(editsCount));
fixup.close();

// End edit list atoms
streamHandler.endElement();
streamHandler.endElement();
```

Figure 4. Code example for sending an edit list for a track in Apple QuickTime movies.

Parallel media in QuickTime files are structured as individual tracks. Among other data, tracks also contain so-called *edit lists* describing which portions of the media are to be played for which duration at which rate. They are represented as QuickTime atoms, namely an EDTS atom containing an ELST atom, which in turn contains one or more ELST element entries.

The code in Figure 4 demonstrates how an edit list is generated in the first stage in the pipeline from Figure 3. It also shows how processing can be separated into components - knowledge on the basic representation of QuickTime atoms is encapsulated in the second component and can be extended seperately, e.g. for handling of atoms larger than 255^4 bytes. Resulting data and fixup events for producing valid atom structures are passed on to the serializer, which ignores the markup events, writes all data references to the file and handles fixups. The resulting hexdump from Figure 4 is shown in Figure 5.

This example showed the use of SABRE for the generation of a small portion of a QuickTime movie. The same pipeline is used in the generation of a full-fledged QuickTime movie which can consist of more than 50 different atom types with type-specific fields, a structure several levels deep, relations and dependencies between fields and embedded media descriptors and formats.

Currently, two student theses are underway which are applying SABRE to the generation of file system images conforming to ECMA-119[7] and ECMA-167[6], also known

```
00 00 00 24 65 64 74 73 ...$edts
00 00 00 1C 65 6C 73 74 ....elst
00 00 00 00 00 00 00 01 ........
00 00 0A 40 00 00 00 00 ........
00 01 00 00             ....
```

Figure 5. Hex dump of an Apple Quicktime track edit list with a single edit (duration 0xA40 ticks, begin at 0x0 ticks, playback rate 0x10000).

as ISO 9660 and ISO 13346/Universal Disk Format (UDF), for CD and DVD media. Although the theses have not yet been completed, valid documents for both formats have already been generated. Further results regarding the usability of SABRE for processing modularization are expected and subject of a future publication.

5. Outlook

During development with SABRE 1.0, several starting points for further work have been identified:

- Although currently sufficient for the given use case, the design and current implementation of SABRE currently does not support variable-length placeholders. Currently, the actual position in the bytestream is evaluated directly and returned as a primitive value, without being able to take into account the final length of possibly still open variable-length placeholders.
- The current DataReference interface implicitly specifies both a data source as well as the data field with offset and length. To mark only a segment of this data as part of an element currently requires this DataReference to be split, which is not supported directly and can only be achieved by inefficiently stacking DataReference implementations. Using separate events for passing references to data sources and data fields would enable step-by-step refinement of complex binary structures using multiple components in a SABRE pipeline.
- The presented API can handle octet-aligned data, but does not provide support for bitstreams. Bitstream support is essential for handling non-octet-aligned binary formats such as MPEG-4 Elementary Stream video streams. Initial work for bitstream support has already begun.

6. Conclusions

This paper has presented the SABRE 1.0 API for parsing, transforming and serializing binary-oriented documents. Its implementation provided a sound basis for the original use case, the modularized serialization of compound lecture recordings in binary Apple QuickTime documents. Further work includes the handling bitstreams and non-octet-aligned data.

The approach allows to easily experiment with implemented formats by adding or modifying pipeline components. For the presented use case, this could cover the generation of video podcasts which are structurally very similar to normal QuickTime documents.

7. Acknowledgments

Many thanks to my collegues, especially Guido Rößling who provided feedback and criticism on the paper in various stages. Furthermore, many thanks go to Andreas Hartl for naming the API.

References

[1] *About SAX*. SAX Project. [Online at http://www.saxproject.org/; accessed 15-April-2006].

[2] *Document Object Model (DOM)*. World Wide Web Consortium (W3C). [Online at http://www.w3.org/DOM/; accessed 15-April-2006].

[3] *Events vs. Trees*. [Online at http://www.saxproject.org/event.html; accessed 15-April-2006].

[4] *Extensible Markup Language (XML)*. World Wide Web Consortium (W3C). [Online at http://www.w3.org/XML/; accessed 15-April-2006].

[5] *JSR 173: Streaming API for XML*. Java Community Process. [Online at http://www.jcp.org/en/jsr/detail?id=173; accessed 15-April-2006].

[6] *Standard ECMA-167: Volume and File Structure for Write-Once and Rewritable Media using Non-Sequential Recording for Information Interchange*. Ecma International, 1997. [Online at http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-167.pdf; accessed 15-August-2006].

[7] *Standard ECMA-119: Volume and File Structure of CDROM for Information Interchange*. Ecma International, 1998. [Online at http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-119.pdf; accessed 15-August-2006].

[8] *QuickTime File Format*. Apple Computer, Inc., 2001. [Online at http://developer.apple.com/documentation/QuickTime/QTFF/qtff.pdf; accessed 15-April-2006].

[9] S. Haustein and A. Slominski. *XML Pull Parsing*. [Online at http://www.xmlpull.org/; accessed 15-April-2006].

[10] G. Rößling, C. Trompler, M. Mühlhäuser, S. Köbler, and S. Wolf. Enhancing Classroom Lectures with Digital Sliding Blackboards. In *Proceedings of the 9^{th} Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2004), Leeds, UK*, pages 218–222. ACM Press, New York, 2004.

[11] A. Slominski. *On Using XML Pull Parsing Java APIs*. [Online at http://www.xmlpull.org/history/index.html; accessed 17-April-2006].