Version Aware LibreOffice Documents

Meenu Pandey
Department of EECS
University Of Wisconsin-Milwaukee
Milwaukee, WI 53201-0784, USA
mpandey@uwm.edu

Ethan V. Munson
Department of EECS
University Of Wisconsin-Milwaukee
Milwaukee, WI 53201-0784, USA
munson@uwm.edu

ABSTRACT

Version control systems provide a methodology for maintaining changes to a document over its lifetime and provide better management and control of evolving document collections, such as source code for large software systems. However, no version control system supports similar functionalities for office documents.

Version Aware XML documents integrate full versioning functionality into an XML document type, using XML namespaces to avoid document type errors. Version aware XML documents contain a preamble with versions stored in reverse delta format, plus unique ID attributes attached to the nodes of the documents. They support the full branching and merging functionalities familiar to software engineers, in contrast to the constrained versioning models typical of Office applications.

LibreOffice is an open source office document suite which is widely used for document creation. Each document is represented in the Open Office Document Format, which is a collection of XML files. The current project is an endeavor to show the practicality of the version aware XML documents approach by modifying the LibreOffice document suite to support version awareness. We are modifying LibreOffice to accept and preserve both the preamble and the IDs of the version aware framework. Initially, other functionality will be provided by wrapper applications and independent tools, but full integration into the LibreOffice user interface is envisioned.

Categories and Subject Descriptors

I.7.1 [**Document and Text Processing**]: Document and Text Editing—*document management*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*version control*

Keywords

user collaboration, XML, version aware

DocEng'13, September 10–13, 2013, Florence, Italy.
ACM 978-1-4503-1789-4/13/09.
http://dx.doi.org/10.1145/2494266.2494269

1. INTRODUCTION

Version control systems provide a methodology for maintaining changes to a document over its lifetime as a collaborative team develops it. Typical version control systems, like Subversion [6], Mercurial [5] , and Git [1], provide the functionality for version repository creation, storage and retrieval of versions from a repository and creation of a graph of versions via branch and merge operations. These tools often require access to a central repository or a shared file system to store the versioned data. Experience has shown that it is a useful service for a large technical user base.

However, for office documents, the user base is typically non-technical. Also, while the documents often go through many revisions, they are often standalone objects or are part of small collections. Thus, the overhead of creating and managing a repository is hard to justify.

One approach to track different revisions of a document is to save them by different names that suggest the evolution of the document. In this approach, collaboration can be achieved by manual branching and merging, but this can be a cumbersome and confusing task. Branch and merge functions for office documents can aid users by keeping track of multiple revisions of a document within the document itself, maintaining branch information for the document when multiple authors work on it simultaneously, and later by merging those parallel changes into a unified version when needed. The ability to track changes of a document is important for many official (user manuals, regulatory documents, technical design documents, etc.) as well as for personal documents.

Office document software does provide simple version control, in the form of current version/past-version (Microsoft Office) or linear document histories (LibreOffice). While this support for versions is helpful, it is insufficient for collaboration in large teams because parallel editing is not supported. This can force users to perform manual merges to integrate changes from multiple sources. Some cloud storage systems offer versioning and collaboration support for stored documents but they do not provide the systematic support for collaboration that a version control system does.

We aim to show that the addition of branch and merge functionality to LibreOffice [3] will facilitate collaboration with less manual effort. The first step towards this goal is to convert LibreOffice ODF files into version aware documents and provide the basics of version control support.

In the rest of the paper we introduce version aware approach for xml documents and how it can be implemented for LibreOffice documents. In subsequent sections, we explain the load and save process of LibreOffice document and

how necessary changes for version awareness can be made persistent throughout the LibreOffice document lifecycle.

2. BACKGROUND AND RELATED WORK

Conventional office document programs already support simple forms of version control. Microsoft Word has a "Track Changes " feature that can be viewed as a two version system. When changes are being tracked, there is a notion of the current version and of a single previous version. Differences between these versions are tracked and a user can decide whether or not to accept the changes. LibreOffice also stores textual documents in a compressed archive that holds a series of files that represent a linear document history. Each document is represented in the Open Office Document Format, based on XML [9]. Users can choose different versions if this is needed. Changes can be recorded and authors can accept or reject the changes between versions but it does not provide three-way merge functionality and does not support simulteneous editing by multiple authors. Thus, neither system supports true collaboration because neither one provides any services for merging parallel edits of the same base document. Whether existing LibreOffice version control functions can be used for better change detection is left for future work.

Conventional source code version systems support branch and merge operations with tools like diff3 [2] which assumes that the source material is raw text and that line breaks represent frequent and meaningful delimiters within files. In fact, modern office document systems often store all their content in XML files with exactly two lines: one for the XML declaration and the second for the rest of the content. In this context, meaningful merging of XML content is challenging, because it is difficult to be certain how to match XML element content between two versions. Some systems use approximate signatures [4], while Thao and Munson showed that using unique IDs allows for an efficient merging algorithm [7]. Based on this work, Thao [8] proposed a new Version Aware document framework using the following elements:

1. a special namespace (called "molhado") to separate the versioning information from the application's normal content;
2. a revision history element in a preamble location that holds the version history information in reverse delta format.
3. XML signature elements to prevent users from altering the version data without detection; and
4. a unique identifier attribute for every element of the document content so that changes between different versions can be identified easily.

In a version-aware document, the latest version contains the complete document content, while previous versions can be retrieved by applying a chain of deltas to the latest version. Each sub-element of the revision history element stores information about the edit operations performed in previous versions. The main edit operations are: attribute value update within any element, changes in node sequences, node deletion, node addition, and node name update. This versioning framework also includes an efficient 3-way merge algorithm that requires each XML node to have a unique ID. Unique IDs are important for efficient matching of nodes between versions. If correctly maintained by an editing system, they allow the versioning system to match nodes between versions even when some nodes have undergone substantial transformations. Unique IDs also help to identify conflicts between two versions, which are currently expected to be resolved manually by the authors.

As a version-aware document contains the entire document history, users do not need to interact with any version repository. Thus users will gain the ability to access past versions, especially gaining the ability to recover contents that were deleted in multiple revisions in the past. Also, the system will provide support for authors to work simultaneously on the same sections of the document by creating separate version branches and to later merge their changes. Non-conflicting changes can be merged automatically while conflicting changes will need manual effort.

A first application of this framework was made using the Inkscape SVG editor, via addition of a wrapper application that manages the maintenance of versioning information in Inkscape saved files. This worked well because SVG editors are designed with the expectation that other applications might add namespace-protected content that should be preserved. In contrast, our first attempts to use the version-aware framework with LibreOffice failed because the versioning information was tolerated by the application, but not preserved during a load-edit-save interaction cycle. This problem has forced us to make deeper changes to the implementation of the LibreOffice software so that it can support the version aware preamble and element unique identifiers.

3. APPROACH AND IMPLEMENTATION

It will not be surprising to learn that a production quality office document system uses a complex file representation. A LibreOffice ODF document is a zip compressed archive that contains four XML files: meta.xml, settings.xml, content.xml, styles.xml. The "meta.xml" and "settings.xml" files do not affect the content of the document. The "styles.xml" contains information about the styles used within the document. The "content.xml" file stores the main content of the document including text, pictures etc. Thus, if the content.xml and styles.xml files can accept the four changes described above, then a LibreOffice document will be version aware. Currently, our main focus is applying the four modifications changes to the content file. We are leaving the changes to the style file for future work. The problem, of course, is that LibreOffice was not designed with version awareness in mind. Furthermore, it did not begin its life as an open source project, so documentation is limited.

In LibreOffice, the document is saved as a set of files on disk but is represented by a rather different document model in memory. During the document load operation, an import filter converts the XML files into this document model. Similarly during a save operation, an export filter converts the document model into XML format. Every element and attribute of the content file has a corresponding data structure in the document model. So, any changes made to the content file that are not part of the LibreOffice document model are not saved during the save operation.

Our approach is to write a wrapper application that will read the content file and will add the "molhado" namespace, the unique identifiers, the revision preamble and the authentication signature for every version of the document,

Figure 1: Content.xml file after applying required changes for version awareness. Namespace *molhado* is defined first. New elements of *revision-history* and *signature* are added. Rest of the elements in xml are assigned a *unique ID*.

as needed. Document size increases once all the needed changes are applied to document by the wrapper application. In addition, the core LibreOffice source code has had to be modified to persist this information through a complete load-edit-save cycle. This task is not straightforward because of the particulars of the LibreOffice implementation, which does not exploit inheritance as much as one might hope, especially in the area of element attributes.

When LibreOffice writer loads an existing ODF file, an xmlreader object reads all the XML files and instantiates objects for every element type. These classes are called Import Context classes and have data members that correspond to the attributes of the respective XML element. First, those data members are assigned the corresponding attribute values. Second, based on the context of the XML element type, the corresponding universal network object (UNO) services are called such as - Paragraph, PageCount, TextCursor etc. Each UNO service has properties which are set by the import contexts created earlier. Then all the UNO service objects are stored in memory as a pool of items, which defines the document model. The document saving operation is straightforward. It creates export context classes for all the items in the item pool and then saves them into XML format.

To accommodate the proposed changes we have made the following changes to the LibreOffice code base:

1. All namespaces are stored in a namespace map in LibreOffice. So the molhado namespace has been added to that namespace map.

2. New elements like signature and revision history are preserved during load and save by making new classes for these elements that save their attributes over the lifecycle of the document.

3. Every XML node needs a unique ID that should not change over the lifetime of the document versions. The unique IDs are essential for identifying nodes that have undergone substantial transformations and can still be matched with their original version. Preservation of the unique IDs for every element requires changes to be made at multiple entry points of the XML elements. The main high level elements in the content.xml are fonts, styles, text, table and number where the functionality can be added to store the unique ids of those elements and their children elements. As every XML

element has a corresponding UNO service, the addition of the new ID attribute in each XML element should be accompanied by the addition of a UNO service property as well.

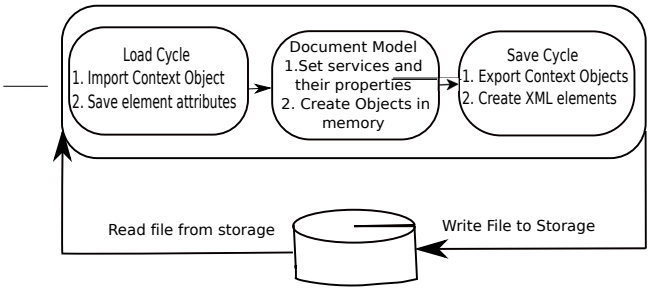

Figure 2: LibreOffice document lifecycle.

As unique IDs should be accomodated for every XML element type, we need to modify the implementation of LibreOffice to support three key operations:

1. Load and save ID attribute in import context classes during XML load operation.

2. Add the ID property to the corresponding UNO services and save IDs of every node in memory.

3. Extract ID property and save it in XML format during export operation.

Once the changes in content.xml are preserved by the LibreOffice application, our wrapper applications will be able to perform three-way XML merging in order combine changes made in parallel by multiple authors.

4. RESULTS

LibreOffice is a long-lived and complex system and determining how to modify it has been a challenging task. We have successfully added our additional namespace and the preamble containing the version history and signature so that they persist through load-edit-save cycles. The key challenge has been to determine the life-cycle of document content elements and to find a good point at which to insert our IDs. At the time of writing, we have been able to make the IDs persist for paragraph elements and for text fields used in forms. But our solution for paragraphs will be quite cumbersome to reproduce for other related elements and we are looking for a more general control point to support the IDs.

The document size increases as additional information is stored in content.xml. File size change for a LibreOffice document with 150 nodes in content.xml is approximately 1.5KB for a first revision when namespace, revision history, ID and signature information is added. It is expected to increase with every revision as incremental revision history will be added. Currently the exact file size change information is not determined because LibreOffice strips off the unique IDs for many xml elements. Thus, versioning framework does not work correctly to calculate change history between versions.

5. CONCLUSION

This application note describes the implementation of the version aware framework for LibreOffice "writer" documents. It also suggests modifications required for the LibreOffice code base so that our changes can persist throughout the document's lifecycle. A version aware LibreOffice document will contain a complete change history and will be able to undergo three-way XML merging and conflict resolution so that document collaboration and management will be possible without the use of a conventional version control repository.

This work shows that it is possible to provide many of the sophisticated features of modern software version control systems in a context designed for less sophisticated users. Branching and merging tasks are already being performed by office document authors, but without adequate automated support. The version aware document approach integrates easily with office document systems because it is designed to work with the XML representation that those systems have already accepted. Thus, full-blown branching and merging can be accessible to non-technical users working on everyday documents.

6. REFERENCES

[1] Git - fast version control system. http://git-scm.com.

[2] GNU diff3. http://www.gnu.org/software/diffutils/.

[3] LIbreOffice. http://docs.libreoffice.org/.

[4] T. Lindholm. A three-way merge for XML documents. In *Proceedings of the 4th ACM Symposium on Document Engineering*, pages 1–10. ACM Press, 2004.

[5] Mercurial SCM. http://mercurial.selenic.com.

[6] Subversion. http://subversion.tigris.org.

[7] C. Thao and E. V. Munson. Using versioned tree data structure, change detection and node identity for three-way XML merging. In *Proceedings of the 10th ACM Symposium on Document engineering*, DocEng '10, pages 77–86, New York, NY, USA, 2010. ACM.

[8] C. Thao and E. V. Munson. Version-aware XML documents. In *Proceedings of the 11th ACM Symposium on Document engineering*, DocEng '11, pages 97–100, New York, NY, USA, 2011. ACM.

[9] Extensible Markup Language (XML). http://www.w3.org/XML/.