Interoperable Document Collaboration

Svante Schubert
Freelancer
Berlin, Germany
Svante.Schubert@gmail.com

Sebastian Rönnau
Freelancer
Berlin, Germany
Sebastian.Roennau@gmail.com

Patrick Durusau
Freelancer
Covington, GA., United States
Patrick@durusau.net

ABSTRACT

To provide office applications with an easy interoperable document merge capability and to enable the usage of document revision across applications, it is necessary to not only standardize the representations of a document state, but also of the changes made to the document during the editing process. Tracking the changes during editing retains the information usually being recovered afterwards. This avoids costly and time consuming processes like document comparison and diff heuristics [1].

To this day, file formats such as the OpenDocument file format (ODF) do only specify all possible document variations of a document representing the final state of user data. Interoperability is therefore only given on a document level: One ODF application saves a document and a different application is able to load and continue work on the same document state. Common scenarios of document exchange have been by floppy disc, attached to email and exchange across networks via file services such as Dropbox.

Nowadays, the Internet is ubiquitous and multiple users want to work simultaneously on the same document. In that context the transfer of a whole document from user to user is inefficient. Additionally, finding and merging changes in XML-based documents appears to be complex and possibly error-prone [2].

For this reason, the OASIS Advanced Document Collaboration subcommittee has started to simplify collaboration by specifying the changes applicable to an ODF document and raising ODF application interoperability from a full document level to a more granular document change level.

In this paper, we present an approach to ODF change representation called "Merge enabled Change-Tracking" (MCT), which is based on the Operational Transformation approach [3].

Categories and Subject Descriptors

I.7.1 [**Document and Text Processing**]: Document and Text Editing;
I.7.2 [**XML**]: Change Control – *merge, change-tracking, versioning;*
C.2.4 [**Computer Communication Networks**]: Distributed Systems – *Distributed applications.*

DChanges '14, September 16 2014, Fort Collins, CO, USA.

ACM 978-1-4503-2964-4/14/09…$15.00.
http://dx.doi.org/10.1145/2723147.2723155

General Terms

Algorithms, Design, Standardization.

Keywords

Real-time collaboration, Merge, XML, Document Changes, Versioning, Operational Transformation.

1. INTRODUCTION

1.1 ODF Basics

An ODF application is any application that is able to load and save a valid ODF document. The ODF document is in general a set of XML files within a ZIP container. A saved ODF document represents the latest state of the user's document. Existing change tracking is implemented by saving the state of a changed region within the XML file before and after the change.

1.2 ODF Change-Tracking Proposals

There is ongoing work at the OASIS OpenDocument format (ODF) Technical Committee to improve collaboration. For this reason, the Advanced Document Collaboration subcommittee (SC)[1] has been created to improve change tracking for the ODF file format.

This task became important as ODF change-tracking is generally considered as underspecified. For this reason, Microsoft has not implemented ODF change-tracking. All Microsoft Office versions remove any tracked changes when loading and saving an ODF document.

There have been three different proposals in the OASIS SC, which we briefly describe in the following.

1.2.1 Generic Change-Tracking (GCT)

In GCT, every change of the XML file is tracked by embracing it with additional change-annotation XML elements, such as "add", "delete", etc., which can be grouped by IDs. GCT has been proposed by DeltaXML Ltd.

GCT aims to be a universal XML change-tracking. The specification is comprehensive yet compact[2]. Despite of that, the documents become large due to high amount of annotation elements resulting from the generic and inefficient encoding. Due to the generic change model, the ODF application model could not benefit from it, as their internal model is in general not ODF XML based at run-time. As result all the generic XML changes

[1] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office-collab

[2] https://www.oasis-open.org/committees/document.php?document_id=41512&wg_abbrev=office-collab

have to be mapped to a semantic user change during the load of a document into an application, mapping the groups of GCT elements to API calls. By this, complexity is moved to the application developer who has to re-identify user change from the many change elements.

1.2.2 Extended Change-Tracking (ECT)

ECT has been proposed by Microsoft[3] [4]. In ECT, a change set representing the previous state of the changed document area is saved in the document and referenced from the changed area. When a change is being rejected, the prior saved XML replaces the area again. This semantically black box approach of changed areas gets problematic in case that multiple changes occur on the same or overlapping areas: if only one of those changes should be reverted, the different areas cannot be clearly distinguished and may influence each other. It is difficult for an application to identify and remove the dependencies of the rejected change from the other overlapping areas of change, as they were only treated as black boxes of ODF XML (same applies for OOXML).

1.2.3 Merge Enabled Change-Tracking (MCT)

In this paper, we present MCT. Its design is based on the idea to first specify the user changes within the ODF model and then mapping that to the XML domain. This moves the complexity out of the documents into the specification, thus storing only a list of the applied user changes within the document. Two open-source early adopters are currently implementing this approach for real-time editing: OX Documents[5] and WebODF[6].

We explain this approach in more detail in the following sections.

A Select Committee voted by the ODF TC resolved in the end the stale mate between the three proposals. All three proposals have been compared and MCT has been chosen for the next version of ODF. The winning factor was its XML coding efficiency for change-tracking: any complex XML change pattern triggered by a user could be mapped to a simple pre-defined operation in the user model, e.g. the move of a row in a large table.

Examples for the different proposals that also exemplify the efficient design of MCT can be seen in the final report[7] of the Select Committee.

2. SPECIFYING DOCUMENT CHANGES

2.1 Design Evolvement

In order to understand the design decision, one must consider the history of browser-based document editing. In most ODF-based collaborative editors known to us, ODF documents are transformed on server side to HTML, sent to the browser on client-side, edited by user and sent back to the server. The server side transformation from ODF to HTML does perform well, but the ability to collaborate on the same document with multiple clients turns out to be difficult. The change by the user is hard to identify in the HTML and even harder to merge.

We would like that the changes may be exchanged by different applications. This is a crucial benefit if emerging browser office solutions should be able to exchange changes with already existing OpenOffice installation, e.g. in companies. This collaboration is desired especially for large documents, making the continuous exchange of the complete ODF document obsolete.

The challenge is to allow two ODF applications with different run-time models to exchange user changes on a document. Both applications have to be able to apply the same user changes to a document, even if not created from the same application.

After any collaboration round, all users have to be able to save the same document. All application models have to provide the same document state. Therefore, the application models have to be kept in sync during collaboration.

To accomplish this, a simplified abstraction layer of ODF has to be used among the applications; state changes are dispatched based upon that high-level unified document model.

2.2 The unified Document Model

We seek for a high probability to find the model in many applications. Therefore, we chose an abstraction model aligned to the human perception of a document. For instance, we describe a position of a change among each other such as: “I have deleted the 3rd character of the 2nd paragraph.” This position we refer to as “/2/3”. The logical components of a document known by all users are being counted and referenced. A component is for instance: an image, paragraph, table (also every contained row and cell) and each character as most fine granular unit.

To refer to a change of a component in an ODF document the position of the ODF XML representing the component is authoritative. A component consists of one or more ODF XML, there is always one single starting element defined for this component – for instance <text:p> for a paragraph or <table:table> for a table. The position of the component is the position of the start element within the ODF document, which is initially being loaded.

2.3 The standardized Change

The ODF standard makes no assumption on the run-time model of an ODF application. Therefore, the application model has to be treated as a black box. It is still sufficient to know that any ODF application is able to load an ODF document, apply a single user’s changes and save the new state as an ODF document.

A certain type of user change is now being defined in the ODF specification by describing the change pattern of the ODF XML occurred between loading and saving the document. Although there is no ODF XML involved at run-time for ODF applications, the applications are able to test their internal state change by comparing the ODF XML input and output.

There are three basic changes types: “add”, “delete” and “modify”. The additional “move” and “replace” operations can be derived from them. The “add” moves any existing component at

---

[3] https://www.oasis-open.org/committees/document.php?document_id=41699&wg_abbrev=office-collab

[4] https://www.oasis-open.org/committees/document.php?document_id=41816&wg_abbrev=office-collab

[5] http://www.open-xchange.com/products/ox-documents

[6] http://www.webodf.org/

[7] https://www.oasis-open.org/committees/document.php?document_id=46485&wg_abbrev=office-collab

this position (and all that follow) one to the back, increasing the position by one.

2.4 Transforming Changes

Any document can be seen as a sequence of changes that has created the document. For instance, systems such as OX Documents map every ODF file on the server to a list of operations, which are sent to their browser office. In return every user action within the browser office results into a change operation.

It is important to realize that different user actions and therefore operations may result into the same document, as a user has different options to create the same document:

The user might type in a sequence of two letters "AB". But the same document might as well be created by inserting first "B" at position 1 and after moving the cursor back to position 1 inserting "A" at the beginning. As both documents are equal, their queues of changes should be able to be transformed into another.

The elemental change is to swap two adjacent operations of the list. Logically this replacement is equal to swap the order of the operations in time.

For example, if someone modified the 3rd paragraph and added afterwards a new 2nd paragraph, swapping the order result into an earlier addition of the 2nd paragraph. Therefore the modification will no longer be on the 3rd as the new 2nd was just inserted. The 3rd was moved back and became the 4th.

This position adoption of an operation is the transformation known as Operational Transformation (OT) [3].

2.5 Relation of Changes

The same change operations can be used for real-time collaboration, change-tracking and undo-redo. Usually the real-time is most fine granular, while undo/redo already more compressed for instance already combines text on word level and change-tracking the changes to component, dropping intermediate changes of the author.

2.6 Inverse Operation

Every operation changing the document from one state to another has also an inverse operation that is able to bring the document back to its original state.

While for real-time collaboration the changes of the users are of importance to be dispatched to other clients, for change-tracking the inverse operation is of importance and will be saved within the document. The combination of both allows the implementation of a history of the document.

2.7 Normalization of Operations

The normalization of operations is important to remove redundant data to efficiently compare documents and changes. The idea is to remove all redundant operations and compress the remaining. To load a document, no prior delete operations are required, the addition & deletion of a component is not changing the document state and can be removed. Operations are sorted in document order and a sequence of multiple text insertions may for instance be combined to one.

Normalization would allow the use of hash algorithms to identify a possible change. Details of this are still under discussion.

2.8 Versioning

There are parallels in versioning of source code in distributed version control systems (DVCS) as via Git[8] or Mercurial and the collaboration on documents. When two users work on the same document without syncing (like offline over the week-end), they work on two branches, which have the last document state as the starting point of their branches. They need to merge their work, before they are able to continue a joint work (on the same branch). Another parallel might be that every document editor might create milestone of their work, similar as doing a commit/version in version control systems.

2.9 Merging of Changes

The collaboration of any number of users can be scaled down to the merge of the changes of two users at the same time.

Two users started their work on the same document, the same queue of operations and added an additional stack of operations to the document.

Merging is similar to DVCS: One of the users (A) is pulling the latest changes from the other (B). The B stack is moved through the A stack as each operation would have been applied earlier than A. The A stack is being adopted by OT becoming A'. Afterwards the end state of user B is equal to the starting state of A', and A' can be simply put upon the operations of B and pushed to B.

2.10 Merge Conflicts

Even the most efficient merge is no prevention from merge conflicts that might occur during asynchronous editing on the same document.

For instance, a user might work on a table cell while the other removes the table. The work on the document might continue while the user decide, which of the two options is applicable.

More dangerous are the semantic errors that cannot be detected without a semantic aware control:

For instance, two users add new first and a new last paragraph, which have no word in common, but still might still have the same meaning, therefore be redundant.

MCT cannot prevent this type of merge errors, but the high-level change representation within the user context may help to identify these errors early.

3. USE CASES OF MCT

3.1 Interoperability of Changes

The change does not have to be identified by diff' heuristics between two documents, as they will be standardized on OASIS/ISO level and provided as a list. Up to now, ODF applications have sent files. With MCT, the defined changes can be dispatched, making a merge and versioning and therefore collaboration far easier across platforms and applications.

3.2 History Function

The ability to keep track of inverse operations allows the ability to go back and forth through the history of a document. It is sufficient to save the final state and all normalized "undo operations" between milestones.

[8] http://git-scm.com/book/en/Git-Branching

3.3 Accept, Reject and now Postpone

Currently, a change can only be rejected or accepted. MCT allows to postpone (aka "cherry picking") a change. This is possible by the ability to move a single or a group of changes through the list of changes representing the document. Making it possible to move changes outside the scope of the final state pointer in the list. This group would be marked similar to a feature branch that has not yet been merged with the main document.

3.4 Merge Efficiency

Aside of the missing identification step of a change, the duration of the merge of operations - representing changes on file documents - is only related to the number of operations, but no longer of the size of the changed document.

3.5 Changing Read-only Documents

As no additional identifiers have to be added to the document to locate the position of a change, but instead the position is being counted by predefined marking elements (the start elements of the components), even changes of remote or read-only (signed) documents can be sent to the owner (similar as in pull-request with DVCS systems as in GitHub).

3.6 Conformance Testing

Every ODF application will be able to be tested for the feature it supports by loading a document, applying the specified change and saving back the document.

Also the ODF documents of a user can be tested on the used feature set, allowing identifying the requirements for an ODF application.

An open-source test suite should be available on OASIS level.

3.7 No Feature Loss

Currently during document import the full state of the document is imported to the run-time model. Any state that does not fit into the model is lost during import.

With the import of operations the features not applicable can be moved out of the import list into a branch, remembered and later during export mapped back to the queue with priority. By doing this, it is possible to have a large gap of feature support among collaborating applications. For instance a text editor (as Vi) could emulate a paragraph with a line and only show character. The Vi could collaborate with an application such feature rich as MS Office on the same full-featured document.

The other option is to read (and sent) only the operations of features known by the other client.

The only difficulty that needs to be addressed is that whenever a client inserts a component at a position adjacent of an unknown component, it is uncertain if the new component is before or behind the unknown component.

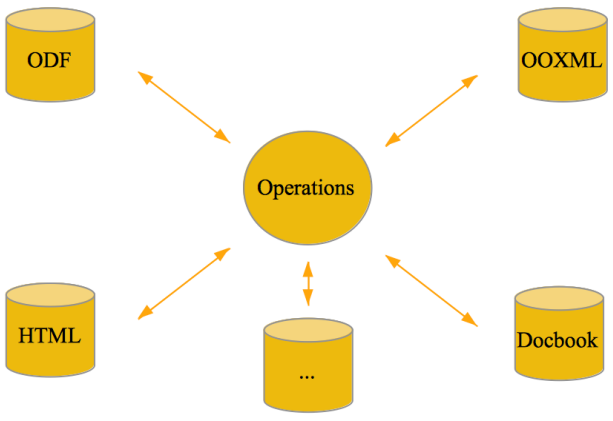

Figure 2. Decouple Transformation Complexity.

If a certain known component, such as a replacement character (character sequence or image), replaces the unknown component, the insertion is determined again.

3.8 Multiple Versions Change

A change like the correction of a typo can be applied to multiple or all branches (versions) of the document where it is applicable. A typical use case is the maintenance of specifications and contracts.

3.9 Replace Semantic

The application could have the ability to track the semantic of global changes. For instance, if a company was renamed the system could give a warning if the old naming will be added in the future again.

3.10 Operations as Lingua Franca

Operations can be used not only to abstract from the run-time models of different applications using the same file format, but as well between similar file formats, like ODF, OOXML, DocBook and HTML.

The available list of operations has to be able to map the feature set of the file format. The browser office OX Documents uses the same operations for ODF and OOXML files. The standardization of ODF changes will certainly be even more successful if there is interoperability to OOXML although the changes were specified in relation of an ODF XML change.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we have sketched a novel approach for change tracking within ODF documents by defining user changes in the ODF specification. It is based on the Operational Transformation (OT) approach and allows for efficient collaboration on structured documents. Our approach, called merge enabled change-tracking (MCT) has been selected as basis for the upcoming ODF standard.

Future work will focus on the detailed specification of MCT, especially the proof of reliable addressing of changes.

A big challenge of the design is the definition of changes (operations) in a way that OT has the lowest complexity when switching two adjacent operations in the changes queue.

First early adopters have already committed themselves to MCT, which will allow us to gain real-life experiences with this approach in order to enable ODF as collaborative document format.

The coverage of both office formats (ODF and OOXML) with as many changes as possible is definitely a goal for future interoperability.

5. REFERENCES

[1] H. Dohrn and D. Riehle. Fine-grained Change Detection in Structured Text Documents. In Proceedings of the 14th ACM symposium on Document engineering, pages 87-96, 2014.

[2] S. Rönnau, G. Philipp, U. Borghoff. Efficient change control of XML documents. In Proceedings of the 9th ACM symposium on Document engineering, pages 3-12, 2009.

[3] Ellis, C. A. and Gibbs, S. J. (1989). Concurrency control in groupware systems. SIGMOD Records, 18(2): 399–407.