ODF Change Tracking Implementation Notes

Notes from the implementation of
generic-ct-proposal
in *KWord* and *Calligra Words*

- Ganesh Paramasivam (ganesh@crystalfab.com)

Table of Contents

1 Introduction

This document provides a short summary of the experiences from the implementation of generic-ct-proposal in KWord and Calligra Words. The topics and areas that are covered by this document are

- Overall approach followed during the implementation.
- Current status of the implementation in KWord and Calligra Words
- Challenges faced during implementation and the details about the same
- Areas where the specification may be improved and extended to address some of the implementation challenges
- Compatibility of the implementation with ODF 1.2 Change Tracking Format

2 Implementation Approach

2.1 Horizontal Task Break-Down

To facilitate an iterative implementation strategy, the task was sub-divided into 4 horizontal broad areas and implementation was followed in the same order as given below.

- Loading of changes
- Saving of changes
- Tracking (from the UI) of changes
- Management of changes (example:- Accept/Reject changes)

2.1.1 Loading Of Changes

This area covers the design and implementation of code to load changes that are available in the proposed ODF Change Tracking format. During the course of implementation of this area, the annotation format for the internal document model was identified that would enable the representation and storage of change information and meta-data.

This implementation co-exits with the ODF loading module of Calligra Words and KWord.

2.1.2 Saving Of Changes

This area covers the design and implementation of the code to save the internal document model containing annotated change information (the format of this annotation was identified in the previous step) to ODF using the mechanisms provided by the proposed ODF Change Tracking format.

This implementation co-exists with the ODF saving module of Calligra Words and KWord.

2.1.3 Tracking Of Changes

This area covers the design and implementation of code to identify and track changes as the document is being edited by the user and to annotate the internal document model with these changes (the annotation format was identified during the implementation of the loading code).

This annotation would then be used by the saving code to save to ODF.

2.1.4 Change Management

This area covers the implementation of User-interface elements to enable the management of changes. Specific functionality implemented during this stage were

- Enabling and disabling tracking of changes
- Enabling and disabling the display of changes
- Format configuration for the display of changes
- Mechanism for enabling accepting and rejecting a specific or a group of changes.

2.2 Vertical Task Break-Down

Vertically all change-tracking use-cases were divided broadly into the following sub-areas and implementation was followed in the same order as given below

2.2.1 Simple Changes (Insertion and Deletion)

1. Text insertion
2. Text Deletion.
3. Full paragraph and header insertions
4. Full paragraph and header deletions
5. List insertions
6. List-item insertions
7. List deletions
8. List-item deletions
9. Inline image insertions
10. Inline image deletions
11. Table insertions

12. Table row insertions
13. Table column insertions
14. Table deletions
15. Table Row Deletions
16. Table Column deletions
17. Paragraph splits
18. Paragraph merges.

2.2.2 Complex Delete merges

1. Paragraph or header merge with a list
2. List merge with a paragraph or a header
3. List item merge with another list item in the same list
4. List merge with another list
5. Paragraph merge with a sub-list in a list.
6. A sub-list merge with a paragraph or header.
7. Sub-list merge with a sub-list in the same list.

2.2.3 Multiple changes

1. Multiple but non-overlapping changes (Some examples of this are).
    1. text insertion followed by a paragraph deletion or a paragraph format change.
    2. List-item insertion followed by a containing List delete.
    3. Table row or column insertion followed by a containing table delete.
2. Multiple and overlapping changes (Some examples are)
    1. Text insertion followed by a partial deletion of this inserted text along with preceding or succeeding original text.
    2. List-item insertion followed by a merge with a previously un-changed list-item.

2.2.4 Format changes.

1. Text format changes (example:- normal to bold, bold to italic etc.)
2. Paragraph format changes (example:- alignment change, background change etc.)
3. List item format changes (example:- numbering re-started etc.)

4. Table cell format changes (example:- Back-ground, Borders, Alignment etc.)

3 Current Implementation Status

The table below shows the current status of implementation for the various change-tracking use-cases in KWord and Calligra Words.

Y indicates that the feature is fully supported

P indicates that the feature is partially supported

N indicates that the feature is not-yet supported.

Use-Case	Loading	Saving	Tracking	Management
		Simple Changes		
Text Insertion	Y	Y	Y	Y
Text Deletion	Y	Y	Y	Y
Paragraph insertion	Y	Y	Y	Y
Paragraph Deletion	Y	Y	Y	Y
List Insertion	Y	Y	Y	Y
List Deletion	Y	Y	Y	Y
List-item insertion	Y	Y	Y	Y
List item deletion	Y	Y	Y	Y
Table insertion	Y	Y	Y	N
Table deletion	Y	Y	Y	N
Table row insertion	Y	Y	Y	N
Table row deletion	Y	Y	Y	N
Table column insertion	Y	Y	Y	N
Table column deletion	Y	Y	Y	N
Inline Image insertion	Y	Y	Y	Y
Inline image deletion	Y	Y	Y	Y

Use-Case	Loading	Saving	Tracking	Management
		Complex Delete Merges		
Paragraph merge with list-item	Y	Y	Y	N
Paragraph merge with sub-list	Y	Y	Y	N
List-item merge with paragraph	Y	Y	Y	N
Sub-list Merge With Paragraph	Y	Y	Y	N
List-item Merge	Y	Y	Y	N
List Merge	Y	Y	Y	N
Paragraph Split	Y	Y	Y	N
Paragraph Merges	Y	Y	Y	N
		Multiple Changes		
Multiple Non-overlapping	Y	Y	N	N
Multiple Overlapping	P	P	N	N
		Format Changes		
Text Format changes	Y	Y	Y	N
Paragraph Format changes	Y	Y	Y	N
List Format Changes	N	N	N	N
Table Format Changes	N	N	N	N
Table-cell merges and splits	N	N	N	N

4 Implementation Challenges and Enhancements

4.1 Challenging Use-cases

One of the challenges faced during the implementation of the proposed ODF Change Tracking format is the way the specification handles and represents certain changes using a combination of low-level mechanisms. More specifically the use-cases for which this problem was encountered were

1. Paragraph or Header merge with list-items
2. List-item merges with paragraphs or headers
3. List-item splits
4. List-item merges
5. List merge with another list
6. Table cell splits
7. Table cell merges

For these above mentioned scenarios, the specification uses a combination of the low-level mechanisms of ***<remove-leaving-content-start>***, ***<remove-leaving-content-end>*** and ***<insert-around-content>*** to specify changes. While this approach results in a precise representation of the changes made to the document, there are two problems with this approach

1. This representation was not the most friendly format to implement and special processing code was needed to process these sections into a much more implementation-friendly formats during loading and saving.
2. The resulting XML format made verification during implementation time consuming.

4.2 Specification Enhancements

The specification can be enhanced to provide mechanisms for the specific handling of these use-cases. The specification already provides mechanisms for the handling of paragraph splits and merges. Since all the challenging use-cases types encountered were either a split or a merge, the same mechanism (paragraph split and merge) can be extended to support the other types of splits and merges as well.

5 Compatibility with ODF 1.2 Change-Tracking Format

The compatibility of the generic-ct-proposal implementation with ODF 1.2 Change Tracking format is given below

- Since the two formats (ODF 1.2 and generic-ct-proposal) are non-conflicting, the implementation is able to load documents containing changes in either of the two formats.
- It is also possible for a document to contain changes in both the formats (since they are non-conflicting). However, the implementation currently does not have the capability to load these documents.
- The format to be used for saving is user-configurable – The user can choose which among the two formats is to be used while saving.
- Since the implementation supports the loading of changes in ODF 1.2 format and the saving of changes in the new format, the application can be used to convert documents from the old format to the new format. However, this conversion has not been tested completely and more work might be needed for to handle certain use-cases (example:- format change markup in ODF 1.2 does not contain the old format, so conversion from ODF 1.2 to the new format is not straight-forward)
- It is also possible to extend the implementation so that the final document may contain changes in both the formats simultaneously thereby making the resultant document both forward as well as backward compatible.

6 Conclusion

In conclusion, based on the experience of implementing the specification, it has been seen that

- The specification is robust enough to handle any type of change that can be encountered.
- It is feasible to implement the specification for all these use-cases.
- The specification is reasonably simple to implement for a majority of the use-cases.
- The final generated XML is human-readable for a majority of use-cases making it simple to verify the same during implementation.

Also, the specification can be further enhanced and extended to provide mechanisms for the support of use-cases identified in section 4 above so that

- The specification is simple to implement for the above specified use-cases.
- The resultant output XML file is easy to verify during implementation.

7 Appendix

7.1 Conformance to Change Tracking Requirements

The table below provides the conformance of the specification against the change tracking requirements identified by the ODF Advanced Collaboration Sub-committee

Y - The specification fulfills requirement.

P – The specification partially fulfills the requirement.

U - Unknown (Not enough experience or information to form an opinion)

Requirements	Status
1.1 Simplicity	P
1.2 Easy to remove	Y
1.3 All changes logged in one place	Y
1.4 Use markup to represent changes not processing instructions	Y
1.5 Can convert from existing change tracking mechanism to new one	Y
1.6 Easy to script	U
1.7 Deleted text stored separately	Y
2.1 Interoperability between different implementations	U
2.2 Interoperability with other formats	U
2.3 Stability of mechanism over long period	U
2.4 Track all possible types of change	Y
2.5 Easy to process with XSLT	U
2.6 Proven implementations	Y
3.1 Future-proof	Y
3.2 Covers all use cases	Y
3.3 High degree of consensus	U

7.2 Building KOffice from Source

7.2.1 Getting Source Code

The source code of KOffice can be obtained by either cloning the git repository or by downloading the tar-ball.

- The git command to clone the KOffice repository is

  git clone git://anongit.kde.org/koffice

- The tar-ball can be downloaded from the following location

  http://anongit.kde.org/koffice/koffice-latest.tar.gz

7.2.2 Installing Dependencies

- **DEBIAN - based distributions (Debian, Ubuntu)**
  All the dependencies used for the packages can be installed by running
  apt-get build-dep {packagename}

  for instance on Ubuntu:
  sudo apt-get build-dep koffice

- **OpenSuSE**
  All the dependencies used for building KOffice can be installed by running
  zypper si -d koffice2

- **ArchLinux**
  All the dependencies used for building KOffice can be installed by running
  sudo pacman -S docbook-xml exiv2 glew graphicsmagick gsl kdebase-runtime kdepimlibs lcms libwpd poppler-qt pstoedit qca qimageblitz shared-mime-info wv2 xdg-utils

7.2.3 Building Koffice

The commands to be used for configuring and compiling KOffice are given below

```
mkdir -p $HOME/kde4/build/koffice
mkdir -p $HOME/kde4/inst
cd $HOME/kde4/build/koffice
cmake -DCMAKE_INSTALL_PREFIX=$HOME/kde4/inst $HOME/kde4/src/koffice
-DCMAKE_BUILD_TYPE = RelWithDebInfo
make
make install
```

NOTE: *The build commands assume that the source-code has been checked-out at $HOME/kde4/src.*

7.2.4 Installing and Running KWord

After a successful compilation of the source code the commands to be used for running KWord are

```
export KDEDIRS=$KDEDIRS:$HOME/kde4/inst
export PATH=$PATH:$HOME/kde4/inst/bin
kbuildsycoca4
kword
```

7.3 Building Calligra from Source

The procedure to be followed to build Calligra from source is documented at the following location

http://community.kde.org/Calligra/Building/Building_Calligra

Please note the change tracking implementation in Calligra is available in the development branch

words-change_tracking-ganeshp