ODF – Advanced Document Collaboration

Svante.Schubert@gmail.com
Freelancer

Status: OASIS OpenDocument - Advanced Document Collaboration SC

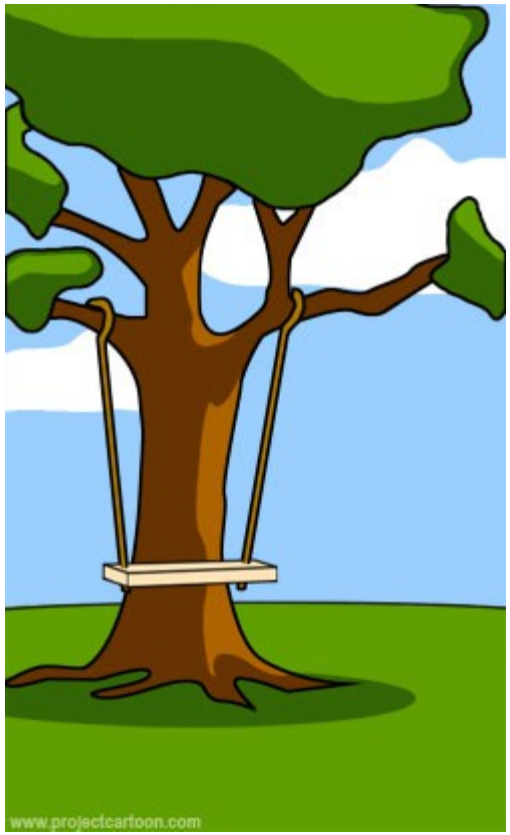
- 1st Proposal: Generic change tracking (by DeltaXML)
  - Tracks every XML change in an ODF document
  - Might be applied to other XML formats
  - Shortest specification, largest ODF documents
- 2nd Proposal: Extended change tracking (by Microsoft)
  - Extending specification of ODF 1.0/1.1 change tracking
  - Saving the changed XML aside
  - Small specification change, small ODF enhancements
- Agreement to focus first on Change Tracking

Mission objective:
Change Tracking

- What is the optimal way to serialize/save run-time changes to the document (ODF)?
  - ODF changes not specified, current CT approaches tracking XML changes
  - No clear optimum as a mathematical derivative of formula to get maximum benefit
  - Important: Follow-up objective of (real-time) collaboration. Can CT solution be extended to fulfill second objective?

Mission objective:
Misunderstanding?

“How customer explained”



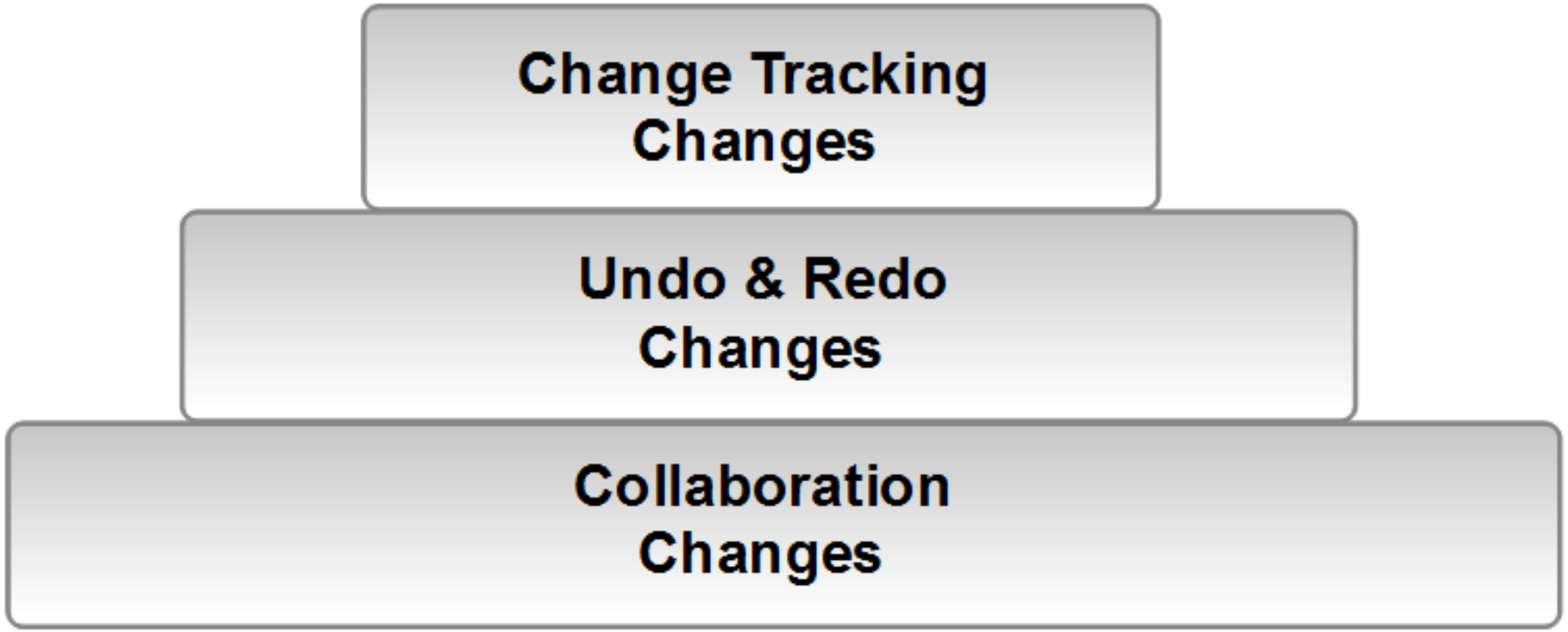
“How project lead understood it”

“What customer wanted”

Mission objective:
Problem Observations

- **Need to merge**
  International group exchange documents via mail.
  Problem of two simultaneous editors, e.g. in EU project
- **Need to save (off-line) Collaboration as CT**
  During collab session Internet access goes down, save changes, synch later
- **Need to map changes between different model of ODF apps**
  Exchange changes between browser and ancient office
- **Need to map CT to human readable changes**
  ISO Errata document: Listing the changes being made in a separate document

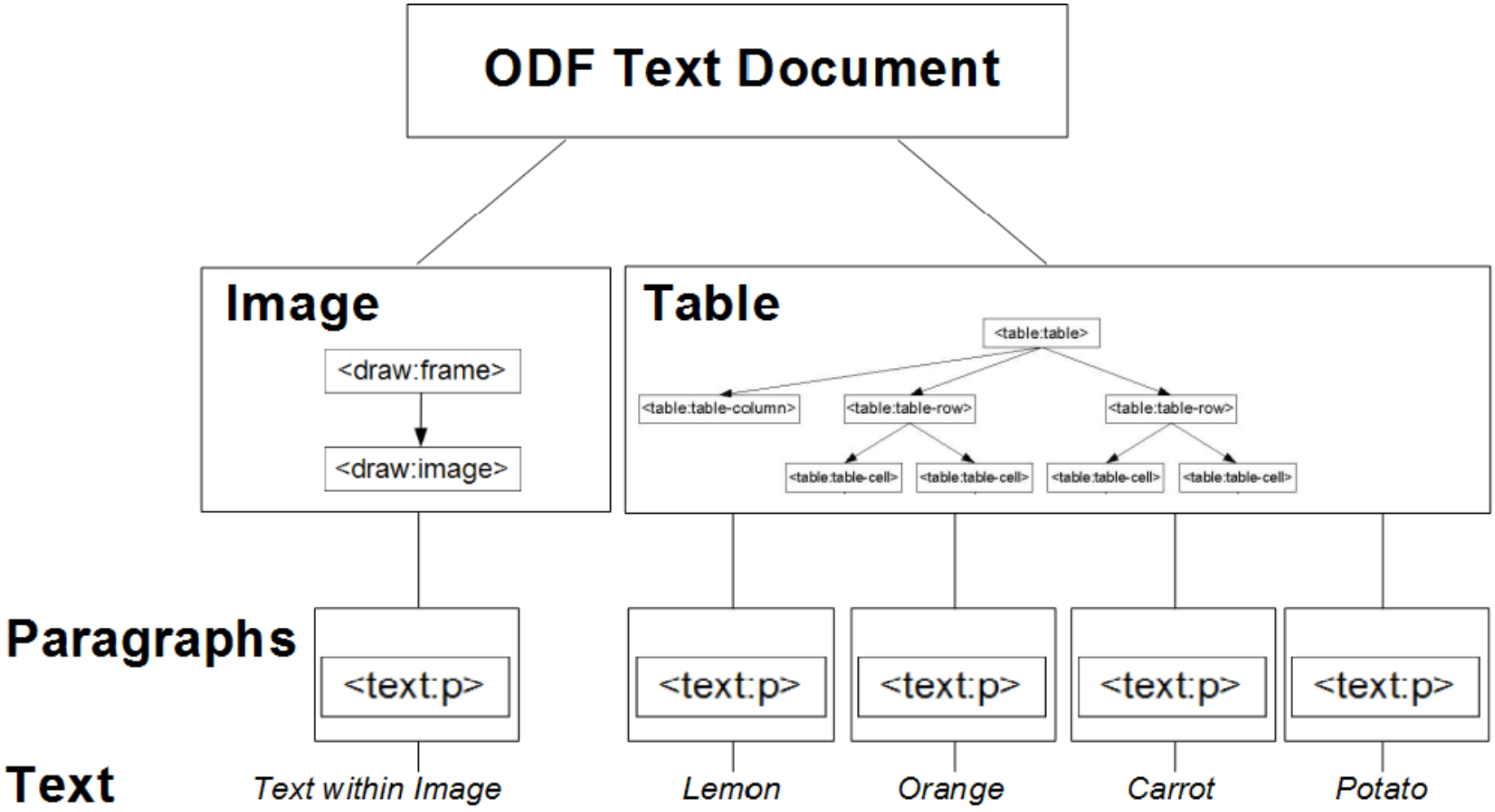
Dependencies of Changes



Collaboration of ODF Applications

- ODF Application: Everything loading/saving ODF
  - For example: Libre|OpenOffice, Browser + JS ODF support
    How can a browser (HTML/CSS model) collaborate at run-time with ancient Offices?
- What is the Lingua Franca between them?
  - All open the same ODF document in a different model
  - ODF Concept: Higher abstraction layer from ODF XML

Document Example with ODF Components



Design - Serializing ODF operations

- Strategic Goal:
  Move repeating patterns of change (complexity) out of documents into spec
- Specifying operation = specifying related XML change
- Label of change referencing to a defined change in spec (e.g. mergeCell, mergeColumn ..)
- Instead of relying on XML, rely on higher level, component tree, where smallest components being characters
- Further convention over configuration: Define empty document, default styles palettes

Design - Serializing ODF operations

- To ease merge operations are referencing ODF components by their relative position
- Serializing queue of ODF operations into single XML file
  - Allow read of changes only (read only “undo.xml”)
  - Allow commenting signed ODF document and singning user changes for each user
- Every operation has an inverse operation
  - add(“paragraph”, “/1”, “Hello World”) <=> del(“paragraph, “/1”)
  - Status of previous ODF XML + do operation(s) =
    Status of current ODF XML =
    Status of upcoming ODF XML + undo operation(s)

Serializing Example: Adding Text & Paragraph

Starting markup:

```
<text:h>Some text.</text:h>
<text:p>Existing important text!</text:p>
```

Ending markup:

```
<text:h>Some text. This could be a very long text!</text:h>
<text:p>New text!</text:p>
<text:p>Existing important text!</text:p>
```

Changes (only FYI - not being saved):

```
<redo>
   <add type=”text” s="/1/11">This could be a very long text!</add>
   <add type=”paragraph” s="/2">New text!</add>
</redo>
```

Undo changes (undo.xml):

```
<undo>
   <del type=”paragraph” s="/2"/>
   <del type=”text” s="/1/11" e="/1/33" />
</undo>
```

Serializing Example: Using Template Styles

Starting markup:

```
<text:p>Normal normal bold bold bold-italic bold bold normal normal.</text:p>
```

Ending markup:

```
<text:p>Normal normal <text:span text:style-name="bold">bold bold <text:span
text:style-name="bold-italic">bold-italic</text:span> bold bold</text:span>
normal normal.</text:p>
```

Changes (only FYI - not being saved):

```
<redo>
   <add type=”style” name=“bold“ s="/1/15" e="/1/46" />
   <add type=”style” name=“bold-italic“ s="/1/25" e="/1/36" />
</redo>
```

Undo changes (undo.xml):

```
<undo>
   <del type=”style” name=“bold-italic“ s="/1/25" e="/1/36" />
   <del type=”style” name=“bold“ s="/1/15" e="/1/46" />
</undo>
```

Serializing Example: Adding/Removing a List with List Items

Starting markup:

```
<text:list>
  <text:list-item><text:p>Line 1</text:p></text:list-item>
  <text:list-item><text:p>Line 2</text:p></text:list-item>
  <text:list-item><text:p>Line 3</text:p></text:list-item>
</text:list>
```

Ending markup:

```
<text:p>Line 1</text:p>
<text:p>Line 2 added</text:p>
<text:p>Line 3</text:p>
```

Changes (only FYI - not being saved):

```
<redo>
   <del type=”list-level” s="/1" e=”/3”>
   <add s="/2/7"> added</add>
</redo>
```

Undo changes (undo.xml):

```
<undo>
   <del s="/2/7” e="/2/13" />
   <add type=”unordered-list” s="/1" e="/3" />
</undo>
```

Serializing Example:
Text Selection, Deletion & Merge

Starting markup:

```
<text:h>Some text. This could be a very long text!</text:h>
<text:p>New text!</text:p>
<text:p>Existing important text!</text:p>
```

Ending markup:

```
<text:h>Some important text!</text:h>
```

Changes (only FYI - not being saved):

```
<redo>
   <del s="/1/6" e=”/3/10”>
   <merge s="/1" />
</redo>
```

Undo changes (undo.xml):

```
<undo>
   <split s="/1/6" />
   <add s="/1/6">text. This could be a very long text!</add>
   <add s="/2">New Text!</new>
   <add s="/3/1">Existing </add>
</undo>
```

Simple Merge Technique

- Operational Transformation (OT)
  - Technical paper of the 80th
  - Google Docs/Wave Protocol build upon
  - Composition / Aggregation:
    E.g.: Several character insertion can be provided as one text
  - Transformation:
    Synching operations from several clients & server
    Check for insertion/deletion of preceding siblings
  - NEW: Using abstraction for better interoperability.
    Instead OT based on XML, based on ODF components

Simple Merge Technique

- Transformation Example:
  - Two clients and server starting on empty document
    - Client A is calling: add(“text”, “/1/1”, “Hello”)
    - Client B is calling: add(“text”, “/1/1”, “ World!”)
    - Client A gets first to server, same version, no problem, server overtakes command
    - Client B gets to server, OT detects & resolves conflict by adapting incoming parameter => add(“text”, “/1/6”, “ World!”)
    - Server overtakes & broadcast to client A
      add(“text”, “/1/6”, “ World!”)

Serializing Example – History Feature Base: OT of Operation when moving within CT Queue

Starting markup:

```
<text:h>Some text. This could be a very long text!</text:h>
<text:p>New text!</text:p>
<text:p>Existing important text!</text:p>
```

Ending markup:

```
<text:h>Some important text!</text:h>
```

Undo changes (undo.xml) - Version A:

```
<redo>
   <split s="/1/6" />
   <add s="/1/6">text. This could be a very long text!</add>
   <add s="/2">New Text!</new>
   <add s="/3/1">Existing </add>
</redo>
```

Undo changes (undo.xml) - Version B:

```
<undo>
   <split s="/1/6" />
   <add s="/1/6">text. This could be a very long text!</add>
   <add s="/2/1">Existing </add>
   <add s="/2">New Text!</new>
</undo>
```

Further Design Details on Serialization Design

- Changes of ODF XML files result in an undo.xml file per document
- Changes will be aligned/referenced to user and time metadata
- Each (embedded) document has its own change-tracking - to easy add/remove doc from ODF package
- Due to tracking 3rd party data (e.g. binary images) there should be a directory ".undo" to store all changed data
- Aside the ".undo" directory there might be a ".redo" directory, in case the document was rolled back
- When allowing a tree of versions those directories might be nested – ongoing research..

Low hanging Juicy Fruits

- Collaboration across ODF applications
  - e.g. OOo|LibreOffice & ODF Browser Office
- Merge functionality
  - Applying changes to remote documents
  - Traffic Light for user after merge (Green, Yellow, Red)
- Arbitrary change grouping
  (and moving changes through time)
  - e.g. ODF Errata version > Contributor > Issue > Change
  - Moving changes through document time-line
  - Access/modify changes independent of (signed) content
- History functionality
  - Saving document as a previous version(s), with ability to restore

Possible next Steps

- Finding the root elements of ODF components. Providing RelaxNG schema that annotates those root elements
- Providing typed properties as intermediate step for each component
- Name methods with typed parameters to change above properties. Including if necessary conditions to guarantee validness.
- An English description of the ODF XML change triggered by any method as new part of ODF specification