Tracked Changes: Navigating the Document-Format Anti-Pattern

Dennis E. Hamilton
Interoperability Architect
4401 44th Ave SW
Seattle, WA 98116, USA
+1-206-779-9430
dennis.hamilton@acm.org

ABSTRACT

Editing of word-processing documents at the presentation level, with visible tracking of changes, operates at a different level of abstraction and granularity than the recorded form in common document-file formats. The consequent mismatches along with other limitations of standards for document-file formats present an anti-pattern that impedes reliable inter-product exchange of change-tracked documents. Analysis of the situation for ODF change-tracking reveals simple extensions and definitions that supplement the current specification without introducing any conflicts. Patterns of systematic testing for conformant document files, compliant processing, and verifiable interoperability are identified as essential prerequisites to dependable improvement of change-tracking in collaborative settings.

Categories and Subject Descriptors

H.4.1 [**Information Systems Applications**]: Office Automation – *word processing;* D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *portability; restructuring, reverse engineering, re-engineering;* D.2.9 [**Software Engineering**]: Management – *life cycle;* H.1.2 [**Models and Principles**] User/Machine Systems – *human factors*, I.7.1 [**Document and Text Processing**]: Document and Text Editing – *version control, document management*

General Terms

Algorithms, Design, Management, Standardization, Verification

Keywords

OpenDocument, XML, change tracking, document formats, overlapping markup, user conceptualization, WYSIWYG.

1. INTRODUCTION

Difficulties in the interchange of documents having tracked changes using document files in open-standard formats is explored using the OpenDocument format (ODF) [10]. Analysis of barriers to interchange begins with the absence of attention to the user-presented document in standard file formats (section 2). Specific disparities are evident in the document-file features for tracked changes (section 3). Reconciliation of the disparities sufficient for improved interchange is then proposed (section 4).

Bridging the experience of users to the technicalities of tracked changes appeals to a notion of document manifestation (section 2.1). This supports a formulation that grounds the repair approach and establishes how adherence can be validated (section 4).

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

ACM 978-1-4503-2964-4/14/09…$15.00
http://dx.doi.org/10.1145/2723147.2723153

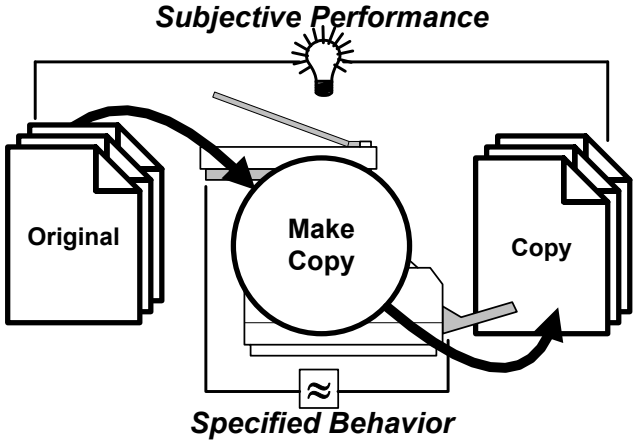

Figure 1. Copying paper fixations of documents illustrates the distinction of purposive use from technical achievement.

2. THE SITUATION

2.1 Terminology: Manifest Equivalence

Office- and personal-productivity software for operation with documents involves coordination between human practice and reliance on technological apparatus whose detailed operation is largely hidden. Appraising the bridge between such disparate levels of abstraction involves unusual nomenclature. Use of a copier illustrates the perspective applied here (Figure 1).

Although it is common to speak of the original and copy as the same document, consider that it is not the case that either of them is *the* document. Instead, regard the copies as each being *fixations* of a document on a physical medium. The fixations afford evocation of the document in the observer's perception.

Consider, then, that what is perceived in the copies are *manifestations* of the document. To the extent that manifestations evoked in comprehension of the original and of the copy are regarded as the same, it can be said that the copy is manifestly faithful to the original.

Construction and operation of the device are specified in terms of technical details far removed from the level at which manifestations are perceived. Details are *engineered* to achieve a confirmable degree of *manifest equivalence* between originals and their copies.[1] Specified behavior is not at that level and fitness-for-purpose of desired performance can be limited.

The "≈" of specified behavior signifies achievement of manifest equivalence under prescribed technical conditions. The domain of validity for manifest equivalence corresponds to an *envelope* within which specified performance is sustained. Validity is contingent, requiring assessment by testing.

A quality of successful engineering is that wherever manifest equivalence is achieved, the underlying technology becomes invisible. Anything else happening within the envelope of specified behavior is a breakdown.

The movement to digitally-originated documents that need not ever have physical fixation on a human-perceivable medium complicates this arrangement in significant ways.

2.2 User Context: Manipulated Manifestation

In today's office-productivity software suites, document production is accomplished by manipulations of a graphical-interface presentation that sustains an ephemeral manifestation of the digital document under development or review. In effect, the operator initiates, manipulates and reviews the manifestation. Success is observation of a manifestation having acceptable appearance of the desired content, reinforced by obtaining a faithful printed edition—fixation—or some final electronic form (Figure 2). Through these interactions, users are trained to their tools through trial and error.

Software that operates the presentation and interaction process maintains, in some manner, an internal document model that captures what there is of the digital document at any point in time. The software also saves work results in one or more document file formats for digital interchange of editable documents. Commonly-employed formats include ODF [9], OOXML [3], and older formats such as those of Microsoft Office [6].

It is important to appreciate that the emitted file formats are not directly used for authoring, editing, or any purpose but re-introduction into computer programs that re-manifest the digital document in humanly-comprehended form. It is near-inconceivable that users of the software have either interest or means to interpret complex document files and discern their relationship to the recognized manifestations that are, for them, the documents. The document files are as opaque to users as are the internal workings of the software. So long as the document-file representation and internal models remain invisible in manifestation-preserving operation, the purpose of the arrangement is achieved.

2.3 Focus of Document Format Standards

The goal for open public standards is achievement of interoperability. Non-interoperable solutions, including so-called *de facto* standards, do not require promulgation of public standards nor the energy and time required to produce, honor, and sustain

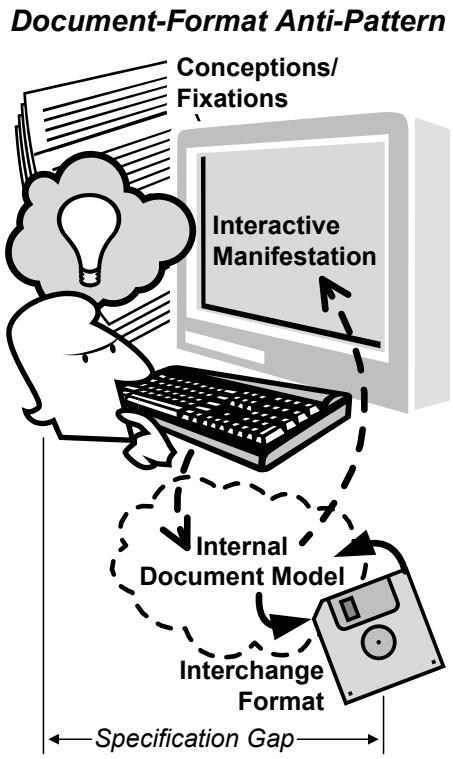

Figure 2. Document processing software implements an internal document model in order to manifest views manipulated at the graphical user interface, shielding users from the intricacies of interchange formats. Microsoft Word, OpenOffice.org Writer, and other personal-productivity software exhibit this approach. The anti-pattern arises when format standards are insufficient to assure preservation of an essential level of manifestation fidelity in the interchange of document files among format-compliant products.

them. The avowed objective is interoperable use of multiple products and ability to make substitutions.

Open standards for office- and personal-productivity software file formats are relatively recent.[2] It is early days and so far standards have at best accidental applicability to coordinated manifestation via document-file interchange. It is nevertheless commonplace for users (and developers) to regard favored implementations as archetypical of the standards.

Current standards are about the emitted files, not about the programs that produce them and consume them for various purposes. Attention is dominated by syntactic structure of the files as recorded data [3][9]. The specifications are addressed to the structure of a document file as it may be encountered, not what gives rise to it nor what is the expected manifestation (Figure 2).

[1] Manifestation is useful in characterizing observable gaps between the worlds of device/software adopters and of device/software producers. Objective reality is not claimed [13].

[2] Although Standard Generalized Markup Language (SGML) and Open Document Architecture (ODA) were innovative International Standards of the 1980s, standardization for office-productivity file formats was not until commencement of OpenOffice XML standard development in December 2002. OpenDocument Format (ODF) 1.0 then emerged as an OASIS standard in May 2005 and as an International Standard in December 2006. In contrast, manual markup of plain-text files for formatted digital presentation arrived with the dawn of general-purpose digital computing.

2.4 Unspecified Interoperability

It is not required that any producer of complex document-file formats be able to emit all features of the format. There is no such claim for any standard-compliant producer of document files in this class. Even so, it is appropriate to expect that software that consumes as well as produces a standard-conforming document-file format can consume what it produces to the satisfaction of the document creator. Failures of this minimum case are legitimately claimed to be software defects.

There is also no minimum requirement on the extent to which features encountered in a given file are to be supported by a *different* conforming consumer of that file (e.g. [10] section 2.4),

> *"[A consumer] shall be able to parse and interpret OpenDocument documents … but it need not interpret the semantics of all [XML] elements, attributes and attribute values. …*
>
> *"It shall interpret those elements and attributes it does interpret consistent with the semantics defined for the element or attribute by this specification."*

It is unsafe to assume that different format-supporting implementations are substitutable based on format conformance alone.

2.5 Interpretation Under-Specification

Although specifications require interpretations, when made, to be consistent with specified semantics, such semantics can be missing or insufficient. Again for ODF 1.2 ([10] section 1.2),

> *"Implementation-defined is used in this standard for values or processing that may differ between ODF implementations, but is required to be specified by the implementor for each particular ODF-implementation.*
>
> *"Implementation-dependent is used in this standard for values or processing that may differ … but is not required to be specified by the implementor …"*

And also ([9] section 1.2),

> *"Undefined behavior may also be expected when the standard omits the description of any explicit definition of behavior."*

The use of "undefined behavior" is unclear unless it is simply read as "interpretation is implementation-dependent wherever the standard is silent."

2.6 The Document Format Anti-Pattern

Specifications for these document-file formats do not address manifestation (2.1-2.2), only the interpretation of already-produced document files (2.3). There is no standard-driven assurance of interoperability among separate implementations (2.4). There is also insufficient specification on which to arrive at consistent interpretations (2.5).

Frustrated expectation of interoperability and substitutability elevates the pattern to a *document-format anti-pattern*[3], a pattern used repeatedly that frustrates a desired outcome (Figure 2).

That is the pattern. It is by design. The specifications do not determine what users of products can rely on by virtue of their standards-compliance alone. Here, compliance causes neither interoperability nor product substitutability.

In this situation, interchange fidelity and interoperability are dependent, when achieved at all, on supplemental agreements, informal or community-adopted, among parties acting beyond the provisions of the document-file standards.[4] Otherwise, manifestation-preserving interchange is more happenstance than engineered achievement.

Realization of interoperability is complicated by three forms of technical debt [4]. There is technical debt of under-specification to be paid down in any remedial specification. There is technical debt of weakly-interoperable implementations to be paid down to achieve dependable interoperability. There is further technical debt concerning means for assessment and validation of conformance and interoperability.

3. ODF TRACKED-CHANGE FEATURES

The OpenDocument Format conveys the structure and content of OpenDocument documents using XML [1]. The provisions for tracked changes at this level determine (and limit) the prospects for manifestation-preserving interchange.

3.1 ODF Change-Tracking Foundation

ODF change-tracking provisions are confined to text documents and spreadsheet documents.[5]

For the change-tracked text documents, the document-file format has an XML `<office:text>` element containing prologue elements, including a `<text:tracked-changes>` element[6]. Following the prologue elements is the text-content flow, a sequence of elements of text-content type, the source for progressive manifestation of the document's layout and content.[7] The over 30 text-content elements consist of elements for tables, lists, headings, paragraphs, document sections, and a variety of shape elements for material to be interspersed in the formatted layout, such as text frames having their own text-content flow along with images and graphics having captions and text labels.

There are no character sequences directly in the text-content flow. Instead, strings of character codes appear interspersed in paragraph-content flow along with elements of paragraph-content type. Several text-content elements (not limited to the paragraph element) contain paragraph-content flow. Some of the over 100 paragraph-content elements can also nest paragraph-content flow and even text-content flow. The schema allowance for recursive content-flow containment is remarkably generous.

---

[3] http://en.wikipedia.org/w/index.php?title=Anti-pattern&oldid=612745076

[4] OOXML does not escape the anti-pattern although there's encouragement of beyond-the-standard third-party application agreements. ([3] part 1, section 2, Conformance). The approach here is along similar lines.

[5] Spreadsheet change-tracking is devoted to transformations of spreadsheet rows, columns, and individual cells and is not considered here.

[6] Individual declarations of page header and footer formats have `<text:tracked-changes>` elements that apply only to change-tracking in those declarations, separate from the `<office:text>` element.

[7] The reading direction of content flow, including through sequences of characters, is always the same as the progression of the XML file stream. Writing directions (both horizontal and vertical) in the manifestation of text fragments are directed by layout controls in that content flow.

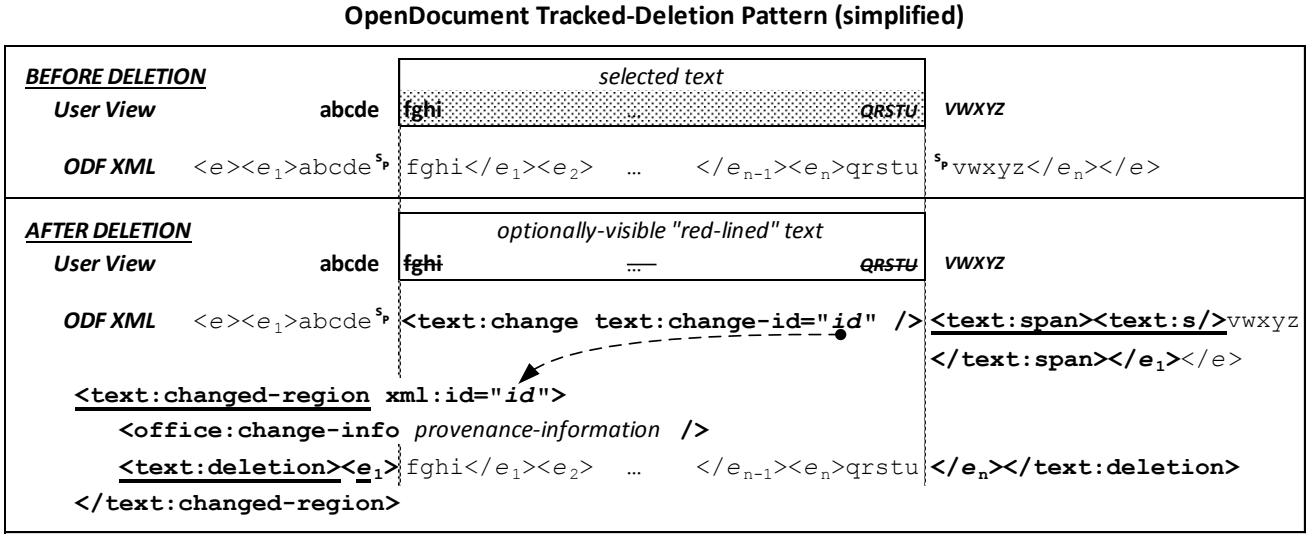

Figure 3. Based on tests using ODF, the user views of a run of formatted text before and after deletion are shown. Receiving document processors only have the after-deletion XML, with no account for how <text:change> and its <text:changed-region> came to be. Compatible receivers synthesize manifestly-equivalent after-deletion views, identifying what was removed and what remains at the level of manifestation. On deletion rejection, receivers synthesize before-deletion XML using material of the <text:changed-region> <text:deletion> element, estimating where excised material was "wrapped" with XML start and end tags in order to be well-formed. Heuristics will determine that the <text:deletion> <e1> can merge with the <e1> fragment before the <text:change>, then blending the orphaned </e1> fragment beyond the <text:change> into the <en> at the end of the <text:deletion>, achieving well-formed XML in the offered restoration of deleted material. Receivers cannot determine that <text:span> and <text:s/> were not present before (here silently introduced at deletion time to preserve <text:change>-adjacent text on deletion acceptance). This case could be remedied by introducing attributes that distinguish those deletion-introduced start elements.

Change marks are the three empty elements <text:change/>, <text:change-start/> and <text:change-end/>. Change marks are text-content and paragraph-content elements. The difference for paragraph-content is the possibility of change marks in the midst of character-code sequences.

Change marks identify where the *consequences* of changes appear. Details about anything removed or altered by those changes are set aside in <text:tracked-changes> <text:changed-region> elements linked from the change marks. Details include provenance data about when the change was made and by whom along with optional comments.

If a change is rejected, information linked from the change marks is used to restore the changed portion of the content flow to its previous form, as necessary.

A <text:change/> occupies the position of a deletion and its <text:changed-region> <text:deletion> holds the excised material in a form for presentation of change-tracking and for reversing the deletion.

<text:change-start/> and <text:change-end/> marks occupy positions where an insertion started and where one ended, respectively.[8] The start-end pairs that identify edges of the same insertion are linked to the same <text:changed-region> and its <text:insertion> component.

It is useful to regard change marks as seams that remain after material on opposite sides of incisions in a content flow have been spliced together. Splicing at seams can involve significant adjustments to adjacent content-flow. Reversal of changes can involve re-splicing material that's restored to the content flow (Figure 3). Adjacent seam-adjustment activity is almost exclusively implementation-dependent.[9]

3.2 Tracked-Deletion Representation

In the XML representation, deletions may be far more consequential than what is manifest and what is evident at individual <text:change/>s in the file. Multiple XML elements of the content flow can be excised. Other elements may have been severed at the deletion-edge incisions, with start tags losing their end tags and *vice versa*. The deletion will appear in the deepest element into which the beginning of the deletion cuts. Splicing may adjust nearby XML tags, introducing additional elements, even character codes, to preserve the manifestation of seam-adjacent content[10]. Deletion is not a context-free activity at the XML representation level.

[8] For simplicity, use of start-end change marks to bracket format changes is omitted in this portrayal.

[9] The entire treatment of change tracking in the ODF 1.2 specification consists of fewer than 60 lines of text ([10] section 5.5).

[10] Since the seam is invisible in manifestation of any paragraph-content character sequence it is within, there are exotic cases where involved-text appearance also changes depending on altered character positions in words, writing-direction changes, and applicable text-joining rules [12]. Manifestation of tracked changes might present these cases as substitutions so the situation is made explicit, comprehensible, and correctible by users.

When XML elements are severed by deletion, the excised material is adjusted by addition of necessary XML start tags and end tags so it is well-formed in the `<text:changed-region>` `<text:deletion>` element (Figure 3). There are no stated limitations on what can be excised and the amount of adjustment involved. The schema allows any text-content flow, including change marks, in the adjusted `<text:deletion>`s.

3.3 Cross-Cutting Markup Effects

Deletions can cross-cut elements and hierarchy depending on where the initial and ending incisions are made (as in Figure 3).

Paragraph-content elements include off-hierarchy bracketing's for non-change-tracking purposes (e.g., annotations and phrase indexing). Any bracket pairing, including of start-end change marks, can be divorced by a deletion that captures one or the other. Deletions can also capture markers that serve as targets (such as bookmarks), as sources for cross-references, and as connections that bind together structural features of the document, breaking the associations.

None of these prospects are constrained or addressed in the specification.

3.4 Atomicity Failures and Copy Conflicts

In ODF 1.2, all changes are reflected in the document file as unconnected insertions and deletions. Substitution by insertion into a selection becomes a deletion and adjacent insertion. If a selection is moved from another part of the document, there is no association of the insertion with deletion at the original location. Failure to treat these actions as joined and atomic leads to conflicts when a deletion is reverted and an associated insertion is accepted. This is exacerbated when implementations divide single-selection deletions and insertions into multiple, smaller change-tracked operations, presenting users with changes not recognizable as theirs and sometimes puzzling in what their acceptance/rejection consequences are.

Copying of certain selections, if allowed, threatens conflicts in the document structure, including violation of internal identifier uniqueness in the XML [5]. The operation, however it is adjusted to prevent conflicts, can result in unintended and over-looked alteration of document-structure interconnections

4. REPAIRING ODF CHANGE-TRACKING

The Repaired Change Tracking (RCT) project is proposed for resolution of the current difficulties by eliminating the under-specification of tracked changes in a manner that achieves down-level compatibility with existing implementations. Development starts with comprehensive analysis of the ODF schema and specification. All interdependencies and interactions between tracked-change markup and other markup of ODF documents are being identified as part of an extensive case analysis and test suite development. Generic cases will be identified and then be expressed in terms of selection, deletion, insertion, copying, moving, and stitching around change marks.

Analysis, collection of details, formulation of a test suite, and development of the profile are carried out on a public web site.[11]

[11] http://nfoworks.org/rct/

4.1 Profile Specification

RCT is being developed as a third-party supplement to the OASIS ODF 1.2 specification [9]. Interoperability is fostered by introducing strict conformance conditions on interpretations of ODF 1.2 and RCT extensions [8], resulting in a profile that restricts and extends ODF 1.2 document files while preserving ODF 1.2 conformance.

For manifestation achievement and preservation, the profile appeals to a fictitious document processor,[12] illustrating what compliant manifestations must afford, not literally how they are to appear or to be accomplished.

Profile provisions establish the envelope within which RCT-conformant document files and compliant processors achieve manifestation-preserving interoperability.

The profile will provide for progressive levels of envelope expansion, with a minimum, basic level detailed first.

4.2 Test Assertions and Test Suites

Test assertions included in the profile are employed in assessment that RCT provisions are satisfied by document files and RCT-compliant processors [11]. Test assertions are not tests. Rather, test suites contain many individual test cases that provide cases of assessment with respect to one or more test assertions.

Test suites are organized so that successful testing is evidence for confidence that the RCT envelope is supported. The objective is to establish a level of measurability that any party can employ in demonstrating and assessing RCT compliance of document files and processors [2].

Testing cannot be exhaustive and cannot prove that an implementation is correct.[13] A feasible level of coverage confirms regularities that are taken as evidence that a great number of similar cases are processed correctly. There are also test cases for assessing how non-conforming document files are treated. Principles of the Isartor Test Suite[14] and the PDF/UA Reference Suite[15] apply.

Conducting tests generally involves manual inspection and appraisal of test-case results, since the comprehension and confirmation of manifestations is necessarily a human activity. Some formalisms and technical procedures are helpful in expediting test case definition and administration.

4.3 RCT Manifestation Fidelity

Let *d* signify a complete document file in ODF format,

x identify a configuration of software, hardware, and user interface, and

$\mathbf{M}_x d$ designate manifestation via *x*.

[12] Blue Sky Office, http://nfoworks.org/bso/

[13] "Program testing can be used to show the presence of bugs, but never to show their absence!" Edsger W. Dijkstra, p.7 in EWD249 (1970), http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF.

[14] http://www.pdfa.org/2011/08/isartor-test-suite

[15] http://www.pdfa.org/publication/pdfua-reference-suite/

From a user perspective, the manifestation is what is perceived. The document file, *d*, is what would be emitted at that instant, signified by

$$\mathbf{M}_x d \rightarrow d.$$

When the document file is re-manifest via the same configuration, the pattern is

$$\mathbf{M}_x d \rightarrow d \rightarrow \mathbf{M}_x d'.$$

RCT Reflection Principle. An RCT-compliant configuration provides faithful manifestation of (RCT-conformant) document files that it produces.

Consequently, $\mathbf{M}_x d$ and $\mathbf{M}_x d'$ are manifestly equivalent, symbolized $\mathbf{M}_x d \approx \mathbf{M}_x d'$.

It is not required, nor is it usual, for files *d* and *d′* to be identical. Nevertheless, differences are insignificant in respect to manifestation equivalence. Under conditions of manifest-equivalence, the files are said to be *manifestation-preserving*, an equivalence relation symbolized $d \cong d'$.

Although manifestations are not mathematical objects, the document files are formal entities amenable to computational treatment. Procedure *alike*(d_1, d_2) compares two document files and assesses whether $d_1 \cong d_2$, reporting `definitely` when that is the case, `definitely-not` when that is definitely not the case, and `indefinite` when the heuristic assessment makes no definite determination. Similarly, procedure *rct*(*d*) reports whether file *d* is assessed to be RCT-conformant using the same result cases as for *alike*. Tuning the procedures is by reducing any bothersome `indefinite` cases while preserving soundness of the definite determinations to within the envelope of specified performance.

4.4 RCT Compliance Assessment

RCT Static Stability Principle. The RCT-compliant manifestation of an RCT-conformant document file manifests the interpreted features of the document file as preserved, without alteration. It is as if the file is read-only or digitally-signed and the goal is for the manifestation to be as close to what the creator observed as possible within the capabilities of the RCT-compliant processor. The purpose is to support at least visual inspection of faithful manifestation and provide for clear-cut test cases.

Individual test cases are of two forms: simple and editing. The simple case involves a single reference document. The editing case involves two reference documents and a procedure specifying the actions that shall be sufficient to transform one to the other.

For simple cases the pair ‹*t*, *f*›, consists of an ODF 1.2 document file, *t*, and a digital fixation, *f*, of the intended document in a form that does not require an ODF processor for its presentation. The fixation might be a raster-image file, a Portable Document Format for Archiving (PDF/A) file,[16] or other fixed digital form whose presentation can be in printed form and affords a proper manifestation of document essentials.

RCT compliance is assessed for configuration *x* and document-file *t* when it is evident that

$$t \rightarrow \mathbf{M}_x t' \text{ is such that } \mathbf{M}_{\text{rct}} f \approx \mathbf{M}_x t'$$

with $\mathbf{M}_{\text{rct}} f$ signifying the manifestation of *f*, in the format employed, as if it were provided by an RCT-compliant processor.

For editing cases the triple ‹*p*, ‹t_1, f_1›, ‹t_2, f_2›› consists of two simple cases and a script, *p*, that conveys a procedure to be performed on a manifestation of the first that results in manifestation of the second using a compliant processor.[17] Editing in this manner is symbolized on the pattern

$$\mathbf{E}_x p(\mathbf{M}_x d_i) \rightarrow \mathbf{M}_x d_{i+1}.$$

Consideration of internal models is avoided entirely. It is the manifestation that is viewed as edited. Alteration of the understood document file is a side-effect, materialized when the file is emitted by the software.

Configuration *x* is assessed compliant for the given editing case if it is evident that, given

$$t_1 \rightarrow \mathbf{M}_x t_1',\ t_2 \rightarrow \mathbf{M}_x t_2',\ \mathbf{M}_{\text{rct}} f_1 \approx \mathbf{M}_x t_1',\ \mathbf{M}_{\text{rct}} f_2 \approx \mathbf{M}_x t_2',$$

$$\mathbf{E}_x p(\mathbf{M}_x t_1') \rightarrow \mathbf{M}_x t_2'' \text{ and } \mathbf{M}_x t_2'' \approx \mathbf{M}_x t_2' \text{ follow.}$$

4.5 RCT Interchange Confirmation

The simplest interoperability case is the ability to make a manifestation-preserving round-trip of document files between configurations *x* and *y*.

RCT Semantic Minimalism Principle. An RCT-compliant configuration shall not produce document files having provisions that the configuration does not interpret (in the sense of section 2.4). This prevents uninterpreted document-file features being passed-through as if recognized and reflected in manifestations when that is not the case.

Round-trip confirmation can be by attempting to achieve

$$\mathbf{M}_x d \rightarrow d \rightarrow \mathbf{M}_y d' \rightarrow d' \rightarrow \mathbf{M}_x d'', \text{ such that } \mathbf{M}_x d \approx \mathbf{M}_x d''.$$

This arrangement detects cases where $d \ncong d'$ and $\mathbf{M}_x d \not\approx \mathbf{M}_x d''$. It is vulnerable to the *works-for-me syndrome*: $\mathbf{M}_x d \not\approx \mathbf{M}_y d'$ not being detected. For interchange testing, it is important to deliver a simple test case, $\mathbf{M}_x d \rightarrow$ ‹*d*, *f*›, rather than just *d*. Fixations are also important in confirming defects and in trouble-shooting interoperability failures.

4.6 RCT Change-Tracking Confirmation

The important case for change-tracking is with interchange of document files where an edited version is returned to the originator. The pattern is

$$\mathbf{M}_x d_1 \rightarrow d_1 \rightarrow \mathbf{M}_y d_1';\ \mathbf{E}_y c(\mathbf{M}_y d_1') \rightarrow \mathbf{M}_y d_2 \rightarrow d_2 \rightarrow \mathbf{M}_x d_2'$$

with *c* signifying some procedure consisting of tracked-change edits. In practice, successful tracked changes are accepted *prima facie*, with no stronger confirmation that $\mathbf{M}_y d_2 \approx \mathbf{M}_x d_2'$ nor that $\mathbf{M}_x d_2'$ differences from $\mathbf{M}_x d_1$ are entirely tracked-change attributable.

[16] http://en.wikipedia.org/wiki/PDF/A

[17] Although it would be handy, the procedure need not be automated.

RCT Indelibility Principle. Tracked changes, even when accepted or rejected, are not automatically removed. Acceptance and rejection are themselves changes, and they are preserved and reversible for auditing and forensic purposes. Change-tracking history and the provenance information are preserved.

Indelibility permits creation of procedure *reversion*(d_2, d_1) yielding d_3, a derivative of d_2 in which every tracked-change that is not present in d_1 is reverted completely.[18] Now $d_1 \cong d_3$ can be assessed. Slim prospect of $\mathbf{M}_y d_2 \approx \mathbf{M}_x d_2'$ works-for-me syndrome remains. When suspected, that and other discrepancies can be investigated using editing test cases as part of trouble-shooting and defect identification.

4.7 Example: Works-For-Me-Not

Attention to validation of manifestation-preservation must deal with the serious works-for-me-not case where the reflection principle is satisfied and the emitted document file is incorrect. That is,

$$\mathbf{M}_x d \rightarrow d' \text{ such that } d \ncong d'.$$

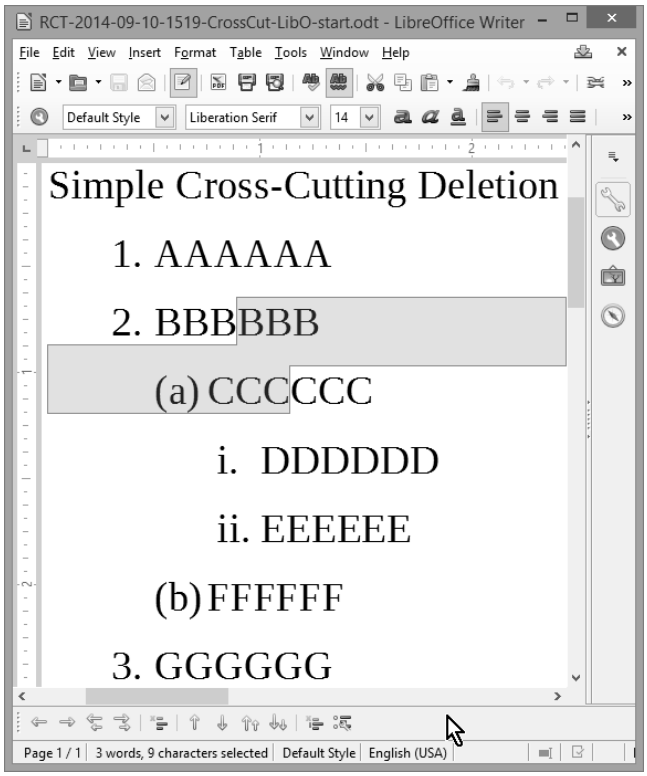

Figure 4. Typical Cross-Cutting Selection Manifestation.

A current works-for-me-not defect occurs when the selection illustrated by the shading (Figure 4) is deleted, resulting in a change-tracked manifestation (Figure 5), where the strike-throughs represent deleted content.

2. BBB~~BBB~~

(a) ~~CCC~~CCC

i. DDDDDD

ii. EEEEEE

(b) FFFFFF

3. GGGGGG

Figure 5. Straight-Forward Change-Tracked Deletion.

On saving the document, however, the form in which the changes are tracked in the document file is manifest as in Figure 6.[19] Although the alteration is presented in the manifestation at the instant the saved document file is created, there is no assurance that alteration is observed by the user at that moment.[20]

2. BBB~~BBB~~

3. ~~CCC~~CCC

i. DDDDDD

ii. EEEEEE

(b) FFFFFF

4. GGGGGG

Figure 6. Altered Deletion on Saving the Document.

Subsequent rejection of the changes in the quietly-altered document will result in the text shown as stricken being simply restored as not deleted. Acceptance is more peculiar (Figure 7)[21].

1. AAAAAA

2. BBBCCC

i. DDDDDD

ii. EEEEEE

(b) FFFFFF

3. GGGGGG

Figure 7. Result from Acceptance of Altered Deletion

RCT resolution is by treating the original selection as two atomically-connected deletions for BBB and CCC. For non-RCT implementations, the two deletions would be seen as decoupled and the works-for-me-not situation avoided there as well.

4.8 RCT Extension Technique

Technical extensions for RCT consist of additional attributes in out-of-line `<text:changed-region>` elements and their change-mark-specific sub-elements. Attributes provide details for correctly-reversing stitching at seams and re-stitching the result (as for Figure 3). `<text:changed-region>` elements are chained together via additional attributes when they are intended to be taken as part of a single atomic change action, with

[18] Since tracked-change acceptance and rejection are entirely in terms of document-file transformation, this is a feasible procedure that can always produce a result such that there are no false-positive determinations.

[19] 2014 releases of LibreOffice and Apache OpenOffice retain this defect that appeared at least as far back as the 2009 OpenOffice.org 2.4.3 release.

[20] The loss of label (a) in Figures 6-7 is accurate.

[21] So far, Microsoft Office has not provided tracked changes in ODF documents it produces. However, the counterpart of Figures 4-5 is recorded by Word 2013 in OOXML as three changes the acceptance of which manifests equivalently to the result in Figure 7.

acceptance or reversal as a whole. The additions are foreign attributes, permitted by the ODF standard for OpenDocument Extended Documents. Implementations may ignore the extended attributes, and the result will be no different than it currently is.

The RCT specification of the manifestation of insertions and deletions is tied to the notion of selections made in the text flow as a consequence of allowed selections in the manifestation. A `<text:changed-region>` is expected to be tied to a selection in the manifestation, there being limited ways to point and select or insert in a practical manifestation.

Selections of sources for copies, moves and their substitutions can be modeled as if selections are extracted into XML markup akin to set-aside `<text:deletion>`s. Insertion of such a copy can be considered akin to but not identical to reversion of a deletion that was already at the place of insertion. Although movement of a selection out of the context of the document in which it is made raises additional challenges, representation of a selection as an XML element in this manner may also be useful for delivery by clip-board negotiation into a different document or for embedding via OLE [7].

5. CONCLUSIONS

Dependable and verifiable interoperability for documents effected by tracked changes is a prerequisite for the success of any approach to tracked changes in standard document-file formats such as those for ODF and OOXML.

Any approach to extend change-tracking of ODF-based documents must address the same manifestation, conformance, compliance, and interchange conditions as RCT. The RCT analysis will be invaluable support to efforts of greater ambition.

Three features of OpenDocument document-file format secure the opportunity for incremental repair and improvement of textual change-tracking via RCT specifically: (1) empty-element XML markers where changes have been made, (2) correct final form when markers are ignored, and (3) extension attributes ignorable by default.

By using selection as an analogy, RCT avoids assumption of specific manifestations of tracked changes and their implementation details. Attention to contingent manifest fidelity provides for demonstration without knowledge of implementations and internal models.

Development and maintenance of provisionally-sufficient test suites is essential for dependable introduction of RCT, demonstration of implementation compliance, and confirmation that there are no deleterious consequences of processing RCT-conforming documents by RCT-unaware down-level software.

6. ACKNOWLEDGMENT

Thanks to the anonymous reviewers for their comments and effort and to the participants in the DChanges 2014 Workshop for their spirited and inspiring discussions. Thanks to colleague William L. Anderson for his enthusiastic attention and careful eye.

7. REFERENCES

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau (editors). *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C Recommendation 16 August 2006, edited in place 29 September 2006. http://www.w3.org/TR/2006/REC-xml-20060816

[2] Jacques Durand (editor). *Interoperability Guidelines*. OASIS Guidelines & Best Practice, 2012-01-14. Available at https://www.oasis-open.org/policies-guidelines/interoperability-guidelines

[3] Ecma International. *Office Open XML File Formats*, 4th edition. Standard ECMA-376, December 2012. http://www.ecma-international.org/publications/standards/Ecma-376.htm

[4] Martin Fowler. Technical Debt Quadrant. Web site article, 2009-10-14. Accessed 2014-10-03 at http://martinfowler.com/bliki/TechnicalDebtQuadrant.html

[5] Jonathan Marsh, Daniel Veillard, and Norman Walsh (editors). *xml:id Version 1.0*. W3C Recommendation 9 September 2005. http://www.w3c.org/TR/2005/REC-xml-id-20050909

[6] Microsoft Corporation. Office File Formats. Microsoft Developer Network MSDN Library. Accessed 2014-06-19 at http://msdn.microsoft.com/en-us/library/cc313118(v=office.12).aspx

[7] Microsoft Corporation. OLE Background. Microsoft Developer Network MSDN Library. Accessed 2014-06-18 at http://msdn.microsoft.com/en-us/library/19z074ky.aspx

[8] OASIS. Guidelines to Writing Conformance Clauses. 2014-04-25 draft update, OASIS Technical Committee Handbook (on-line). Accessed 2014-10-03 at http://docs.oasis-open.org/templates/TCHandbook/ConformanceGuidelines.html

[9] OASIS. *Open Document Format for Office Applications (OpenDocument) Version 1.2*. 29 September 2011 OASIS Standard. master document introducing further parts by reference, http://docs.oasisopen.org/office/v1.2/os/OpenDocument-v1.2-os.html

[10] OASIS. *Open Document Format for Office Applications (OpenDocument) Version 1.2 Part 1: OpenDocument Schema*. 29 September 2011 OASIS Standard. Available at http://docs.oasis-open.org/office/v1.2/os/

[11] OASIS. *Test Assertion Model Version 1.0*. 15 October 2012 OASIS Standard. All forms available at http://docs.oasis-open.org/tag/model/v1.0/os/

[12] The Unicode Consortium. The Unicode Standard Version 5.2.0 defined by: *The Unicode Standard, Version 5.2* (Mountain View, CA: The Unicode Consortium, 2009. ISBN 978-1-936213-00-9). http://www.unicode.org/versions/Unicode5.2.0/

[13] Terry Winograd and Fernando Flores. *Understanding Computers and Cognition*. Addison-Wesley (Reading, MA: 1987). ISBN-0-201-11297-3.