diffi: diff improved; a preview

Gioele Barabucci
Cologne Center for eHumanities, Universität zu Köln
gioele.barabucci@uni-koeln.de

ABSTRACT

`diffi` (diff improved) is a comparison tool whose primary goal is to describe the differences between the content of two documents regardless of their formats.

`diffi` examines the stacks of abstraction levels of the two documents to be compared, finds which levels can be compared, selects one or more appropriate comparison algorithms and calculates the delta(s) between the two documents. Finally, the deltas are serialized using the extended unified patch format, an extension of the common unified patch format.

The produced deltas describe the differences between all the comparable levels of the inputs documents. Users and developers of patch visualization tools have, thus, the choice to focus on their preferred level of abstraction.

CCS CONCEPTS

• **Applied computing** → **Version control**; • **Software and its engineering** → **Software configuration management and version control systems**; • **Information systems** → *Document representation*; • **Human-centered computing** → *Collaborative and social computing*;

KEYWORDS

diff, content comparison, format-agnostic document comparison

ACM Reference Format:
Gioele Barabucci. 2018. diffi: diff improved; a preview. In *DocEng '18: ACM Symposium on Document Engineering 2018, August 28–31, 2018, Halifax, NS, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3209280.3229084

1 INTRODUCTION

It would be great to export a Google Sheet to a local disk as a CSV file, manipulate it with local tools and then upload it again to Google Docs while preserving the changes made to the remote Sheet while we were working locally. At the moment this cannot be done. The fundamental reason why this cannot be done is that these two documents (the Google Sheet and the CSV) are not directly comparable in practice. We regard their content as comparable, but this content is encapsulated in these documents in a way that current diff tools cannot deal with.

DocEng '18, August 28–31, 2018, Halifax, NS, Canada

ACM ISBN 978-1-4503-5769-2/18/08…$15.00
https://doi.org/10.1145/3209280.3229084

There are many examples of documents that are sort-of-comparable but that, in practice, require fragile preprocessing steps, usually embedded in ad-hoc applications:

- HTML and ODT documents;
- XML and JSON files;
- PNG and TIFF files;
- Protobuf and Thrift records;
- C source code in different indentation styles.

`diffi`[1] (*diff improved*) is new tool whose aim is to describe the differences between the content of two documents, regardless of their formats.

In order to reach this aim, several theoretical and practical issues have to be addressed:

- how can documents that use different formats be compared
- at different levels of abstractions
- using a diff algorithm that is appropriate for that combination of documents and levels of abstraction?

In this moment, `diffi` is a research instrument, rather than a production tool. Through its development, three different theses are going to be proved (or disproved):

(1) that the CMV+P document model (Content, Model, Variants and Physical level) and its concept of stacks of abstraction level [4] are sound, valid and useful abstractions;
(2) that all 2-document diff algorithms can be described in three phases: a) structural alignment phase, b) changes detection phase and c) delta refinement phase;
(3) that in a diff tool the code that implements the comparison algorithm(s) is much smaller than the surrounding code that deals with deserializing the documents and serializing the delta into a patch file.

Only the first of these theses will be discussed in this brief article.

`diffi` provides a significant improvement in the way comparison algorithms are developed, improved and studied. By using `diffi` as their base framework, the authors of comparison algorithms are free to focus on the algorithms and the associated data structures, without the need to reimplement all the gritty details that a diff tool must care about.

2 OVERVIEW OF `DIFFI`

`diffi` takes in input a pair of files (source and target), calculates their differences and returns a patch file that describes which modifications must be done to the source file to turn it into the target file.

One peculiarity of `diffi` is that multiple abstraction levels are compared, not only one as in on other tools. For instance, the traditional GNU diff tool compares files as a list of newline-delimited strings of bytes, regardless of whether they are ODT, C++ or PNG

[1] The source code of `diffi` is available at https://gioele.io/diffi.

files. Instead, `diffi` reports if and how the abstraction levels of the two documents differ from one another. For example, in the case of two ODT files, diff will report differences:

- at the "sequence of paragraphs" level (which paragraphs are different? how have they changed? which headings? Have words been moved around?),
- at the XML level (which attributes have changed? which elements? which subtrees has been moved?),
- at the Unicode level (what are the differences between the way the XML files have been serialized? e.g. which code-points have been decomposed?),
- at the UTF/UCS level,
- at the bitstream level.

The fact that `diffi` can provide an overview of what has changed at multiple levels of abstraction addresses two issues: the practical problem of focusing on the right kind of differences, and the philosophical question "are these two documents different?".

The practical problem of showing to the user the right kind of differences is a common, yet unsolved, problem in many fields that make extensive use of diff tools [5, 8]. Take, for example, the case of software engineering. Many practitioners lament the unsuitability of the available tools: sometimes too many details are shown (e.g. lines are shown as different because the amount of whitespace has changed), other times too little information is shown (e.g., how did the signature of this function change? how did the memory layout of this structure change?). The root cause for this problem is the fact that most source comparison tool rely on line-based diffs. Their results are thus related to the textual representation of the source code. In `diffi`, the textual representation of the code is just one of the many abstraction levels that is compared; other levels are, for example, the abstract source tree level, the class architecture level and so on. While comparing source files `diffi` will find the differences at all these abstraction levels and the developers will be able to focus on the abstraction levels they care about.

The philosophical issue of understanding whether two documents are different [10] is addressed by making it explicit that it is possible (and in fact quite common) that two documents are different at a certain level of abstraction while being different at others. For example the same XML tree can be serialized in completely different ways. These two copies of the document would be identical at many levels (e.g. XML-tree level) but different at others (e.g., Unicode level, bitstream level). In this case `diffi` would show exactly that: no differences for the higher abstraction levels and a detailed list of differences for the each lower level.

2.1 Comparison across formats

The fact that documents are seen as stacks of abstraction levels allows `diffi` to compare content across documents that are saved in different formats. Take for instance an HTML page and a flat ODT file, whose stacks of abstraction levels are depicted in Figure 1. Most of the levels at the bottom of these stacks are incomparable and comparing the few that are comparable would lead to meaningless results. At the top of the stacks, instead, there are levels that can be meaningfully compared, for example the "sequence of paragraphs" level. A comparison at that level will produce a list of differences among entities such as "paragraphs", "headings" and "figures", all concepts of that particular abstraction level.

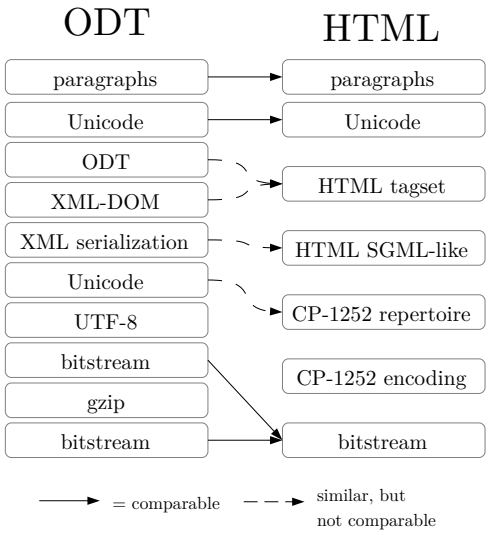

Figure 1: CMV+P stacks for a compressed flat ODT file and an HTML

2.2 Comparison procedure

The comparison workflow in `diffi`, schematically illustrated in algorithm 1, consists of four steps:

(1) decoding the source and target files and understanding which abstraction levels they are composed of,
(2) finding which abstraction levels can be compared,
(3) computing the deltas of each pair of comparable levels,
(4) serializing the deltas produced in the previous step into a patch file.

```
Program diffi file1, file2
    stack1 = LevelsInFile(file1);
    stack2 = LevelsInFile(file2);
    cmp_levels = IndexesOfComparableLevels(stack1, stack2);
    deltas = new Array();
    forall (idx1, idx2) in cmp_levels do
        L1 = stack1[idx1]; L2 = stack2[idx2];
        // L1 and L2 have the same format
        format = FormatOfLevel(L1);
        algo = AlgoForFormat(format);
        deltas « Differences(algo, L1, L2);
    end
    EmitPatch(deltas);
end
```

Algorithm 1: `diffi`'s comparison process

During the first step, the source and the target files are deserialized and their respective stacks of abstraction levels are built.

The result of this step are two stacks, each containing a number of abstraction levels, each of which has an associated model (e.g. XML) and a set of typed content elements (e.g. nodes, processing instructions). The concept of abstraction levels and the CMV+P model behind them are described in more detail in Section 3.

In the second step, the levels in the two stacks are pairwise analyzed to find out which levels of the source stack can be compared to which levels of the target stack. Two levels can be compared if they share the same model (a more formal definition of comparability is given in Section 3.1). The result of the second step is a list of pairs of indices that indicate which pairs of levels of the source and the target document can be compared. For the example in Figure 1 the list of comparable levels would be (0,0), (2,0), (9,6), (10,7).

The third step is the one in which the content of the comparable levels is compared. The comparison is done using an algorithm that is appropriate for the levels being compared, for example the Myers' algorithm [9] for sequential content or Faxma [7] for XML trees. The user can choose other algorithms, for example JNDiff [6] for literary documents. This step does not produce a single delta like other diff tools. Instead, multiple deltas are created: one for each pair of comparable levels. The deltas are stored in data structures inspired by the Universal Delta Model, briefly discussed in Section 4.

In the fourth and last step, the deltas are serialized into a single patch file. This patch is encoded according to the Extended Universal Patch format described in Section 4.

3 DOCUMENTS AS STACKS OF ABSTRACTION LEVELS

Given how central the concept of abstraction levels is for `diffi`, it is necessary to introduce this concept in more rigorous terms along with the CMV+P document model [4] from which it originates.

In a nutshell, the CMV+P model formalizes the common practice of describing how to serialize a file format in terms of other simpler formats. For example, a flat ODT file is described as an XML document that uses a particular vocabulary. An XML document then is serialized as a series of Unicode codepoints, that, in turn are serialized according to the rules of a UTF-8 and so on, eventually becoming a series of bits. These bits are finally embedded on a physical carrier. Each of these steps is, in CMV+P parlance, an abstraction level. Together they form a stack of abstraction levels.

What follows are more rigorous definitions of the CMV+P document model in its linear version, i.e. a stripped-down version of the full model. This linear version can model only non-aggregate documents (i.e. non zipped files).

Definition 3.1. A **document** $\mathfrak{D}$ is a potentially infinite stack of abstraction levels L_i:

$$\mathfrak{D} = (L_0, L_1, L_2, \dots)$$

Definition 3.2. An **abstraction level** L is a tuple composed of a set of addressable elements C, a reference model M and a set of variants V:

$$L = (C, M, V)$$

Definition 3.3. The **content** of an abstraction level is a set C containing addressable elements and relations between them (e.g. order relations). The kind of elements that can be present in C and their structure are dictated by the model M.

Definition 3.4. The **model** of an abstraction level is a reference M to a specification that describes what are the types of the elements in C and what are the constraints of its structure (e.g. Unicode 7, XML 1.0, HTML 5, CSV as defined by RFC 4180, etc.).

Definition 3.5. The **set of variants** of an abstraction level is a set V containing records of the choices, among those made available by the model M, made during the creation of C (e.g. the order of the XML attributes, normalization forms in Unicode, etc.).

3.1 Comparability, equality, equivalence

Definition 3.6. Two abstraction levels L_a and L_b are **comparable** if and only if they share the same model, i.e. $M_a = M_b$.

Definition 3.7. Two *abstraction levels* L_a and L_b are **equal** if and only if they are comparable and their content sets are identical, i.e. $M_a = M_b$ and $C_a = C_b$.

Definition 3.8. Two abstraction levels L_a and L_b are **equivalent** under the equivalence relation eqv if and only if they are comparable and all the elements that are different in C_a and C_b have associated variants v_a, v_b and these variants are equivalent under eqv, i.e. $\exists\,(c_a, c_b) \in \delta\,(C_a, C_b) \leftrightarrow \exists v_a \in V_a, \exists v_b \in V_b, eqv\,(v_a, v_b)$.

4 THE UNIVERSAL DELTA MODEL AND THE EXTENDED UNIFIED PATCH FORMAT

The `diffi` comparison process produces many deltas, one for each pair of comparable abstraction levels. Each delta describes the differences between two comparable levels using a *model-specific set of operations*. The kinds of elements on which these operations work are also model-specific. For example, at the Unicode level the sequences of codepoints are modified by adding, removing or moving codepoints, while at the XML level a tree is modified by adding an attribute, wrapping a sequence of elements or deleting a subtree. The Universal Delta Model (UniDM) [2, 3] describes in abstract terms how to record all these model-specific operations, from the most basic ones (addition and deletion) to the most complex ones (moving, wrapping, splitting, etc.)

These deltas are, however, abstract entities. To be useful to users, deltas must be materialized, for example as patch files. `diffi` produces patches in a novel format: the Extended Unified Patch (EUP) format. This format is an extension of the classic unified patch format that is currently the de facto standard for the exchange of patch files.

EUP extends the traditional unified patch format in two ways: first, it allows the description of differences at multiple levels of abstractions, second, it defines how to serialize model-specific operations. In comparison, the traditional unified patch format is defined for only one precise level of abstraction (the idiosyncratic "lines of ASCII letters" level of abstraction) and describes only two operations: addition and removal (plus context information).

A EUP file, exemplified in figure 2, is structured as follows:

- an header with the names of the files being compared,
- a sequence of abstraction levels, each of which has:
  - a header with the human-readable name of an abstraction level followed by its numerical ID,
  - a list of hunks, each of which has:
    - position informations about the elements being changed,

* a list of changes to be applied, surrounded by context elements.

The first character of each line identifies the class of the change. If there is only one kind of elements in that particular abstraction level, then the following first-character codes are used:

- a space identifies context elements,
- a + identifies an addition,
- a – identifies a deletion,
- a * (followed by a space and a capitalized name) identifies a model-specific operation,
- a # identifies human-readable explanations of the instruction that follows it.

All these operations identifiers are followed by an ASCII representation of the element or of a pointer to that element.

If the model of the abstraction level being compared allows for more than one kind of elements, then the encoding of the changes is extended to include the type of the element in the following way:

```
<op> <tab> <elem type> <tab> <elem representation>
```

```
--- file.odt
+++ file.html
=== paragraphs / 4.21 ===
@@ -1,1 +1,4 @@
# In the middle of the summer...
 1
# -Álvaro!, cried her...
-2
# The sun was shining furiosly...
 3
# Paragraph "Álvaro!, cried her..." moved here; split
     into "Álvaro!" and "Cried her"
*Split(Move(2,3),6)
# Nobody replied...
 4
=== Unicode / 1.3 ===
[...]
@@ -130,2 @@
# (space)
 0020
#-Á (decomposed as A + acute)
-0041
-0301
# l
 006C
# v
 0076
[...]
@@ +176,1 @@
# (space)
 0020
#+Á (precomposed)
+00C1
# l
 006C
# v
 0076
```

Figure 2: Example of an extended unified patch (excepts). A paragraph has been moved and split. In addition, the source ODT file uses decomposed Unicode accented letters while the target HTML file uses precomposed codepoints

5 RELATED WORKS

There is a plethora of specialized comparison tools for almost any imaginable kind of document and file format: Microsoft Word documents, images, fonts, ontologies, VHDL code, etc. All these tools are limited to the comparison of one particular format and at one particular abstraction. Their internal models and their output formats are almost always format-specific and proprietary, both in the sense of non publicly documented and in the sense of not usable in conjunction with other tools.

De facto, the most interoperable tool is GNU diff. Many of the cited tools do, in fact, internally use GNU diff (or a compatible variant) to produce a patch and then interpret the results of the patch making use of their knowledge of the formats.

A notable example of such extended use of GNU diff is diffoscope [1], an in-depth comparison tool that is able to describe the differences between two archives, for example two Debian packages or two zip files, by comparing not only the archives themselves but also the content of the archives, using format-specific deserialization tools and comparing their textual representations.

Some tools claim to be able to compare the content of the documents, going beyond, for example, small variations due to the export process. In practice, these tools have a pre-processing step that normalizes the content of input files, getting rid of "unimportant" variations. The resulting normalized documents are then compared using a format-specific diff algorithm, but only at a specific level of abstraction.

6 CONCLUSIONS

This brief overview of `diffi` showed the basic workflow, the internal mechanisms and the output format that `diffi` uses to compare the content of documents, even those that are stored in different formats.

While still a prototype, `diffi` has already provided insights in the way the various abstraction levels of which documents are composed can be compared and the kind of analysis that is made possible by the CMV+P model.

REFERENCES

[1] diffoscope: in-depth comparison of files, archives, and directories. https://diffoscope.org/.

[2] Barabucci, G. Introduction to the universal delta model. In *Proceedings of the 2013 ACM Symposium on Document Engineering. Florence, Italy, September 10-13, 2013* (2013), S. Marinai and K. Marriott, Eds., ACM.

[3] Barabucci, G. *A universal delta model.* PhD thesis, Università di Bologna, 2013.

[4] Barabucci, G. The CMV+P document model, linear version. In *Versioning cultural objects.* IDE, 2018. (in print).

[5] Barabucci, G., Ciancarini, P., Di Iorio, A., and Vitali, F. Measuring the quality of diff algorithms: a formalization. *Computer Standards & Interfaces 46* (2016).

[6] Di Iorio, A., Schirinzi, M., Vitali, F., and Marchetti, C. A natural and multi-layered approach to detect changes in tree-based textual documents. In *ICEIS 2009* (London, UK, 2009), vol. 24 of *LNBIP*, Springer-Verlag, pp. 90–101.

[7] Lindholm, T., Kangasharju, J., and Tarkoma, S. Fast and simple XML tree differencing by sequence alignment. In *Proceedings of the 2006 ACM Symposium on Document Engineering, Amsterdam, The Netherlands, October 10-13, 2006* (2006), D. C. A. Bulterman and D. F. Brailsford, Eds., ACM, pp. 75–84.

[8] Munson, E. V. Collaborative authoring requires advanced change management. In *Proceedings of the International workshop on Document Changes: Modeling, Detection, Storage and Visualization, Florence, Italy, September 10, 2013* (2013), G. Barabucci, U. M. Borghoff, A. D. Iorio, and S. Maier, Eds.

[9] Myers, E. W. An O(ND) difference algorithm and its variations. *Algorithmica 1*, 2 (1986), 251–266.

[10] Renear, A. H., and Wickett, K. M. Documents cannot be edited. In *Proceedings of Balisage: The Markup Conference 2009* (2009).