

# Diffing, Patching and Merging XML Documents: toward a Generic Calculus of Editing Deltas.

Jean-Yves Vion-Dury

September 2010

Xerox Research Centre Europe  
6 chemin de Maupertuis, 38330 Meylan, France  
viondury@xeroxlabs.com

## Abstract

This work addresses what we believe to be a central issue in the field of XML diff and merge computation: the mathematical modeling of the so-called *editing deltas* and the study of their formal abstract properties. We expect at least three outputs from this theoretical work: a common basis to compare performances of the various algorithms through a structural normalization of deltas, a universal and flexible patch application model and a clearer separation of patch and merge engine performance from delta generation performance. Moreover, this work could inspire technical approaches to combine heterogeneous engines thank to sound delta transformations. This short paper reports current results, discusses key points and outlines some perspectives.

## 1 Introduction and State of the Art

XML diff operations are basically used to realize two fundamental operations on structured documents: comparison and merging. Therefore tree based diff algorithms are central to solving various important problems related to XML document management and editing processes ([2]), among which:

- checking modifications with respect to a reference document, typically, the last version stored inside a repository or a file system (version management [3])
- synchronizing variants, that is, detection of changes that occurred concurrently on two copies of a reference document, detecting potential conflicts ([6])
- merging variants, which is about fusing two variants into a unique document integrating modifications of both instances (this may imply

solving potential conflicts when the underlying document management system applies a weak control policy)

- recovering anterior state(s) of a document

An abundant literature exists on this topic, including recent and excellent synthetic work on an old topic, linear differencing ([5]). Most work covers :

1. *Algorithmic complexity of the differencing operation, either for ordered or unordered tree models.* The problem was first approached as a particular case of string oriented diff computation (see [5] for a point on most recent algorithms in this area), implying computing an adequate linearization of the XML trees. Zhang and Shasha described a fast algorithm in 1989 for computing tree edit distance [17] improving previous work conducted in 1979. Still the runtime and space complexity was higher than quadratic on sequential architectures, but could drop to quadratic levels on parallel architectures. In [15] an algorithm for unordered tree was presented with a polynomial complexity for optimal differences, whereas previous works established a NP-complete complexity for general cases. A survey (still of interest 14 years later) and a comparative study on the topic can be found in [1] and [4]. Most recent algorithms for ordered diff computation are described in [7], [10, 11].
2. *Optimality of editing scripts (also called deltas).* The very notion of optimality is highly controversial. From a pure theoretical perspective, it means measuring the number of atomic operations (this is a simple transposition of the so called Levenstein distance on strings), however, this method lacks two qualities to be really relevant: (i) a normal form for editing scripts has never been defined and (ii) different algorithms can not be compared using a common vocabulary of deltas. From engineering perspectives, measuring performance depends on the implementation of the delta application engine (far from being objective as depending on languages and implementer's skills), and the code is usually optimized toward particular document profiles and some implicit applicative expectations. In addition, usages and document types deeply impact the structure of deltas and thus the runtime performance of algorithms. As a consequence, diff algorithms should ultimately be chosen and parameterized on the basis of document type and related activities.
3. *Delta models and use of diff operations inside storage architectures (typically databases).* In [9], a versioning graph is proposed based on XLink designation mechanism. But in this proposal, the graph relates two separated documents that may exist in independent storage infrastructure. Hence there is no notion of encapsulation and explicit

consistency. In [13], an application is proposed that offers online versioning services for XML documents. However, there is no notion of history encapsulation, and the history itself cannot be exported or organized into an explicit form (see [12] for an online demo). A recent paper [10] proposes reversible deltas using a straightforward mechanism as deletion operation conveys explicitly the deleted subtree (this could quickly lead to enormous overhead, especially in the perspective of storing the whole history). The authors focus on algorithmic and speed performance issues of a novel diff algorithm using a bottom-up approach (and also precomputed hash codes on a sliding window of adjacent nodes.) Another approach from DB community explored a diff model based on so-called *completed deltas* with strong properties among which reversibility and reduction of delta composition [8]. However, the designation mechanism entirely relies on maintaining persistent identifiers for each node of the tree. Thus, according to this framework, no path are needed and the problem of comparing nodes is reduced to numerical equality (of IDs). However, this approach confines the application field to document management systems based on a dedicated indexation system.

As a conclusion, we observe that most work conducted in this area focused on related algorithm performance, time and space complexity of the diff operation and also optimality of generated deltas, but little attention has been paid to tools and models to make use of these in efficient ways. From the practical point of view, no work considered the necessity and applicative interest of abstracting over deltas and related operation models.

## 2 The formalism: definitions

### 2.1 XML model and basic operations

We propose to use a simplifying data model for capturing XML particularities: total order over nodes including attributes (direct translation of the linear semantics of document) but order over attributes not significant in tree comparison. Moreover, attributes can be attached to element nodes only, and are uniquely defined through a name. We denote by  $d$  the document fragments (or subtrees),  $t$  the terminal nodes (text, attribute value) and  $n$  the names of elements or attributes. The following syntax captures the the positional information of nodes thanks to the exponents. Empty element nodes are captured through  $n^i[\epsilon]$ , just noted  $n^i$ . The following attributed Tree grammar capture our tree model together with most structural constraint of XML.

**Definition 1** *document and tree model*

$$\begin{aligned} d^i &::= n^i[f^k] \mid t^i \mid @n^i = t & (\forall i > 0) \\ f^k &::= f^{k-1}d^k & (\forall k > 0) \\ f^0 &::= \epsilon \end{aligned}$$

At this abstraction level, we assume that the lexical constraints of XML are always satisfied, and so are well-formedness constraints, leading to always successful parsing and identification of documents with trees resulting from parsing. In our proposal, trees are operands of abstract operations (thus, similar to *abstract types* [16]): **diff**, **get**, **patch** and **merge**.

**Definition 2** *Abstract modeling of basics functions*

$$\forall d_1, d_2, \exists \Delta \quad \mathbf{diff}(d_1, d_2) = \Delta \quad [\mathbf{a-diff}]$$

$$\forall d_1, d_2, \exists \Delta \quad \mathbf{patch}(d_1, \Delta) = d_2 \quad [\mathbf{a-patch}]$$

$$\frac{\mathbf{diff}(d_1, d_2) = \Delta}{\mathbf{patch}(d_1, \Delta) = d_2} \quad [\mathbf{p-patch}]$$

$$\frac{\mathbf{diff}(d, d_1) = \Delta_1 \quad \mathbf{diff}(d, d_2) = \Delta_2 \quad \Delta_1 \perp \Delta_2}{\mathbf{patch}(d, \Delta_1; \Delta_2) = d'} \quad [\mathbf{a-merge}]$$

$$\mathbf{merge}(d_1, d_2, d) = d'$$

Note that **diff** and **patch** are far from being defined *stricto sensu*: the formulas a-diff and a-path just state that the operations shall work on any pair of well-formed documents, producing a result  $\Delta$  which syntax will be detailed in the sequel. Patch and diff relate through an invariance property *p-patch*. Finally, the (conflict-free) three way merge requires a logical condition, we called it orthogonality, over the deltas issued from applying diff on  $d_1$  and  $d_2$  with respect to a reference document  $d$ .

## 2.2 Paths

Paths are closely related to trees and serve as designation mechanisms. The definition (3) below shows three types of path to designate the elements, attributes and text nodes in this order, and is similar to many previous works (e.g. [10, 11]), but starting counting children nodes at value 1, and being always relative to what is assumed as being the context tree. These two points are not minor, as they enables algebraic operations on paths (see (5)) and also compositional concatenation.

**Definition 3** *Syntax of paths*

$$\begin{aligned} p &::= pp \mid pp/@n \mid pp[\ell] & (1) \\ pp &::= i/pp \mid i & (2) \end{aligned}$$

This syntax distinguishes paths  $p$  designating all kind of node including terminal text and attributes, and paths  $pp$  designating only element nodes. A path  $p$  is well-formed with respect to a document  $d$  ( $d \models p$ ) if it indeed allows accessing part of  $d$ :

**Definition 4** *well-formed paths and tree access*

$$d^i \models i$$

$$\frac{t^i \models pp}{t^i \models pp[\ell]}$$

$$\frac{n^j[\dots @n^i = t \dots] \models pp}{@n^i = t \models pp/@n}$$

$$\frac{d^j \models pp}{n^i[\dots d^j \dots] \models i/pp}$$

**Definition 5** *tree access*

$$\frac{d = n^i[f^j]}{\mathbf{get}(d, i) = d}$$

$$\frac{\mathbf{get}(d, i) = n^i[\dots d^j \dots] \quad \mathbf{get}(d^j, pp) = d'}{\mathbf{get}(d, i/pp) = d'}$$

$$\frac{\mathbf{get}(d, pp) = n[\dots @n^i = t \dots]}{\mathbf{get}(d, pp/@n) = t}$$

$$\frac{\mathbf{get}(d, pp) = t^i}{\mathbf{get}(d, pp[\ell]) = t^i}$$

**Proposition 1** *Soundness of path well-formedness*

$$d \models p \Rightarrow \exists d' \mid \mathbf{get}(d, p) = d'$$

The proposed paths can be compared (useful for asserting precedence over subtrees) and added (useful to propagate changes in the structure of the tree).

**Definition 6** *reflexive path ordering based on prefix*

$$\forall pp \quad pp \subseteq pp \tag{3}$$

$$\forall pp, p \quad pp/p \subseteq pp \tag{4}$$

This definition expresses subtree embedding. Intuitively,  $1/2/3$  designates a subtree of  $1/2$ , which is directly captured by  $1/2/3 \subseteq 1/2$ . The following relation is larger, and defines a strict path ordering that reflects the total document node ordering defined in the XML data model:

**Definition 7** Strict path ordering.

$$\begin{aligned} \frac{i < j}{i/pp \prec j/pp'} \text{ [}\prec\text{-1]} \quad \frac{i < j}{i/pp \prec j} \text{ [}\prec\text{-2]} \quad \frac{i < j}{i \prec j/pp'} \text{ [}\prec\text{-3]} \\ \frac{pp \prec pp'}{i/pp \prec i/pp'} \text{ [}\prec\text{-4]} \quad \frac{i < j}{i \prec j} \text{ [}\prec\text{-5]} \quad \frac{pp \prec pp'}{pp/@n \prec pp'} \text{ [}\prec\text{-6]} \\ \frac{pp \prec pp'}{pp \prec pp'/@n} \text{ [}\prec\text{-7]} \quad \frac{pp \prec pp'}{pp[\ell] \prec pp'} \text{ [}\prec\text{-8]} \quad \frac{pp \prec pp'}{pp \prec pp'[\ell]} \text{ [}\prec\text{-9]} \end{aligned}$$

Note that by construction, this relation is irreflexive. The next relation, smaller, captures ordering between sibling nodes, and turned out to be central in studying dependencies among delta operations:

**Definition 8** Sibling path ordering

$$\begin{aligned} \frac{i < j}{i \ll j} \text{ [}\ll\text{-i]} \\ \frac{i < j}{i \ll j/pp} \text{ [}\ll\text{-i/pp]} \\ \frac{pp \ll pp'}{i/pp \ll i/pp'} \text{ [}\ll\text{-pp]} \\ \frac{pp \ll p}{pp[\ell] \ll p} \text{ [}\ll\text{-}\ell\text{-L]} \\ \frac{p \ll pp}{p \ll pp[\ell]} \text{ [}\ll\text{-}\ell\text{-R]} \end{aligned}$$

These relations lead us to define a more abstract relation, the orthogonality of paths, which application is to assess the independence of atomic deltas, and subsequently, of composite deltas.

**Definition 9** Path orthogonality

$$\frac{\neg(p \subseteq p') \quad \neg(p' \subseteq p) \quad \neg(p \ll p') \quad \neg(p' \ll p)}{p \perp p'} \text{ [}\perp\text{-path]}$$

Now we introduce a partial path addition as the commutative relation satisfying the following properties:

**Definition 10** *path addition*

$$\begin{aligned}
i/pp_1 \oplus j/pp_2 &= (i+j)/(pp_1 \oplus pp_2) \\
i/pp \oplus j &= (i+j)/pp \\
pp_1/@n \oplus pp_2 &= (pp_1 + pp_2)/@n \\
pp_1[\ell] \oplus pp_2 &= (pp_1 + pp_2)[\ell]
\end{aligned} \tag{5}$$

Path addition is useful to propagate structural information related to tree modification. Together with what we call *fingerprints*, it allows to relocate atomic deltas thus permitting swaps in modification sequences.

**Definition 11** *path fingerprint*

$$\begin{aligned}
\zeta(i) &= 1 \\
\zeta(i/pp) &= 0/(\zeta(pp)) \\
\zeta(pp/@n) &= \zeta(pp)/0 \\
\zeta(pp[\ell]) &= \zeta(pp)
\end{aligned} \tag{6}$$

### 2.3 The syntax of delta-based transformations

We propose to express a clear distinction between atomic operations (noted  $\delta$ ) and transformations of tree using combinations of atomic deltas (noted  $\Delta$ ). Composite transformations are built up using Atomic deltas and three operators:

**Definition 12** *Composite and atomic delta-based Transformations*

$$\Delta ::= \delta \mid \Delta_1 \diamond \Delta_2 \mid \Delta_1 \bowtie \Delta_2 \mid \Delta_1 ; \Delta_2 \tag{7}$$

$$\delta ::= \mathbf{ins}(p, d) \mid \mathbf{del}(p) \tag{8}$$

The  $\Delta_1 ; \Delta_2$  notation expresses sequences of transformations, understood that  $\Delta_2$  is applied on the tree resulting from applying the  $\Delta_1$  transformation. Thus,  $\Delta_1$  and  $\Delta_2$  necessarily refer to different trees. On the contrary,  $\Delta_1 \diamond \Delta_2$  describes transformations that can be done concurrently without any risk of interfering each others. This operator allows describing concurrent transformations, in that sense that whatever sub-transformation is applied first, a subsequent merge is always feasible. Moreover, both sub-transformations use the same tree operand as reference (or equivalently, the structural changes induced by one operation can not modify the references involved in the other operation). As an example,

$$\mathbf{patch}(a^1[b^1[c^1]d^2], \mathbf{ins}(1/1/2, e) \diamond \mathbf{del}(1/2)) = a^1[b^1[c^1 e^2]]$$

and however

$$\begin{aligned} \mathbf{patch}(a^1[b^1[c^1]d^2], \mathbf{del}(1/2); \mathbf{ins}(1/1/2, e)) &= \\ \mathbf{patch}(a^1[b^1[c^1]], \mathbf{ins}(1/1/2, e)) &= a^1[b^1[c^1 e^2]] \end{aligned}$$

as well as

$$\begin{aligned} \mathbf{patch}(a^1[b^1[c^1]d^2], \mathbf{ins}(1/1/2, e); \mathbf{del}(1/2)) &= \\ \mathbf{patch}(a^1[b^1[c^1 e^2]d^2], \mathbf{del}(1/2)) &= a^1[b^1[c^1 e^2]] \end{aligned}$$

The third operator,  $\Delta_1 \bowtie \Delta_2$  lies in between, since commutative up to a transformation of the right-hand delta. This transformation will not be described here, and is basically a transposition of paths in  $\Delta_2$  having a dependency on tree modifications induced by  $\Delta_1$ . This transposition, noted  $[\Delta_2]^{\Delta_1}$  is based on fingerprints, path addition and ordering of paths.

**Definition 13** *Equational properties of deltas*

$$\Delta_1 \diamond \Delta_2 = \Delta_1; \Delta_2 = \Delta_2; \Delta_1 \quad \text{if } \Delta_1 \perp \Delta_2$$

$$\Delta_1 \bowtie \Delta_2 = \Delta_2 \bowtie \Delta_1$$

$$\Delta_1 \bowtie \Delta_2 = \Delta_2; \Delta_1 = \Delta_1; [\Delta_2]^{\Delta_1} \quad \text{if } \Delta_1 \ll \Delta_2$$

**Definition 14** *Sibling dependencies* ( $\star \in \{\diamond, ;, \bowtie\}$ )

$$\frac{p \ll p' \vee p = p'}{\mathbf{ins}(p, d) \ll \delta p'}$$

$$\frac{p \ll p'}{\mathbf{del}(p) \ll \delta p'}$$

$$\frac{\Delta \ll \Delta_1 \quad \Delta \ll \Delta_2}{\Delta \ll \Delta_1 \star \Delta_2}$$

$$\frac{\Delta_1 \ll \Delta \vee \Delta_2 \ll \Delta}{\Delta_1 \star \Delta_2 \ll \Delta}$$

**Definition 15** *Orthogonality of deltas* ( $\star \in \{\diamond, ;, \bowtie\}$ )

$$\frac{p \perp p'}{\delta p \perp \delta p'}$$

$$\overline{\mathbf{del}(p) \perp \mathbf{del}(p)}$$

$$\frac{\Delta \perp \Delta_1 \quad \Delta \perp \Delta_2}{\Delta \perp \Delta_1 \star \Delta_2}$$

$$\frac{\Delta_1 \perp \Delta \quad \Delta_2 \perp \Delta}{\Delta_1 \star \Delta_2 \perp \Delta}$$

### 3 Synthesis and perspectives

The definitions above (mainly, the equational laws) will allow us designing a normalization function, and also designing sound transformations for optimization purpose or e.g. for adapting the delta vocabulary to any specific patch engine. So far our formalism defined the following elements:

1. a simple tree model still capturing basic XML peculiarities (in particular, mixed node ordering policies),
2. a simple designation mechanism to specify tree areas and various items of the tree,
3. a syntax for building diff operations,
4. a compositional semantics of diff operations taking in account operation dependencies.

This last point, based on an axiomatic approach outlined in [14], is not fully presented here. Our next step will be first to assess formally the soundness of our definitions, explore the consistence of our whole system through structural lemmas and:

1. propose a mapping of known constructs into our delta model (update-node, delete-node, delete-tree, insert-node, insert-tree, append-node, append-tree, move, copy ; see [14] for a first approach),
2. propose a normalization process and relevant transformations of deltas,
3. prove soundness of previous operations,
4. show how to model real world applications (patch, merge) using the formalism.

Ultimately, our theoretical framework should help reasoning on architecture optimization, establishing sound metrics to assess qualitative and quantitative performance of algorithms and tools, and last but not least, inspire and help defining novel approaches among which patching algorithms based on streaming execution models.

### References

- [1] P. Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1-3):217–239, 2005.
- [2] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 493–504. ACM, 1996.

- [3] S. Chien, V. Tsotras, and C. Zaniolo. A comparative study of version management schemes for XML documents. Technical Report TR-51, TimeCenter technical Report, September 2001.
- [4] G. Cobéna, T. Abdessalem, and Y. Hinnach. A comparative study for XML change detection. *Research Report, INRIA Rocquencourt, France*, 2002.
- [5] S. Khanna, K. Kunal, and B. Pierce. A Formal Investigation of diff3. *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, pages 485–496, 2007.
- [6] R. La Fontaine. Merging XML files: a new approach providing intelligent merge of XML data sets. In *XML Europe*, pages 03–03. Citeseer, 2002.
- [7] T. Lindholm, J. Kangasharju, and S. Tarkoma. Fast and simple xml tree differencing by sequence alignment. In *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*, pages 75–84, New York, NY, USA, 2006. ACM.
- [8] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet. Change-centric management of versions in an XML warehouse. In *Proceedings of VLDB*, pages 581–590, 2001.
- [9] M. Martínez, J.-C. Derniame, and P. de la Fuente. A method for the dynamic generation of virtual versions of evolving documents. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 476–482, New York, NY, USA, 2002. ACM.
- [10] S. Rönnau, C. Pauli, and U. M. Borghoff. Merging changes in xml documents using reliable context fingerprints. In *DocEng '08: Proceeding of the eighth ACM symposium on Document engineering*, pages 52–61, New York, NY, USA, 2008. ACM.
- [11] S. Rönnau, G. Philipp, and U. M. Borghoff. Efficient change control of xml documents. In *DocEng '09: Proceedings of the 9th ACM symposium on Document engineering*, pages 3–12, New York, NY, USA, 2009. ACM.
- [12] A. Rosado, A. Mrquez, and M. Snchez. A web-based version editor for XML documents. online demonstration at <http://picaro.unex.es:8180/vEditor/>.
- [13] A. Rosado, A. Mrquez, and M. Snchez. A web-based version editor for XML documents. In *Proceedings of the 9th ACM symposium on Document engineering*, pages 249–250. ACM, 2009.

- [14] J.-Y. Vion-Dury. Stand-alone encoding of document history. In *Balisanage: The Markup Conference 2010*, August 3 - 6 2010.
- [15] Y. Wang, D. DeWitt, and J. Cai. X-Diff: A fast change detection algorithm for XML documents. In *International Conference on Data Engineering (ICDE'03)*. Citeseer, 2003.
- [16] Wikipedia. Abstract data type. *en.wikipedia.org*, 2010.
- [17] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.