Chapter II
A Comparative Study of XML Change Detection Algorithms

Grégory Cobéna
INRIA, France

Talel Abdessalem
Telecom ParisTech, France

ABSTRACT
Change detection is an important part of version management for databases and document archives. The success of XML has recently renewed interest in change detection on trees and semi-structured data, and various algorithms have been proposed. We study different algorithms and representations of changes based on their formal definition and on experiments conducted over XML data from the Web. Our goal is to provide an evaluation of the quality of the results, the performance of the tools and, based on this, guide the users in choosing the appropriate solution for their applications.

INTRODUCTION
The context for the present work is change detection in XML data warehouses. In such a warehouse, documents are collected periodically, for instance by crawling the Web. When a new version of an existing document arrives, we want to understand changes that occurred since the previous version. Considering that we have only the old and the new version for a document, and no other information on what happened between, a diff (i.e. the delta between the two versions) needs to be computed. A typical setting for the diff algorithm is as follows: the input consists in two files representing two versions of the same document; the output is a delta file representing the changes that occurred.

In this paper, we consider XML input documents and XML delta files to represent changes. The goal of this survey is to analyze the different
existing solutions and, based on this, assist the users in choosing the appropriate tools for their applications. We study two dimensions of the problem: (i) the representation of changes (ii) the detection of changes.

**Representing changes.** To understand the important aspects of changes representation, we point out some possible applications:

- In Version management Chien et al. (2001), Marian et al. (2001), the representation should allow for effective storage strategies and efficient reconstruction of versions of the documents.
- In Temporal Applications Chawathe et al. (1999), Zhang et al. (2004), the support for a persistent identification of XML tree nodes is mandatory since one would like to identify (i.e. trace) a node through time.
- In Monitoring Applications Chen et al. (2000), Nguyen et al. (2001), Jacob et al. (2005), changes are used to detect events and trigger actions. The trigger mechanism involves queries on changes that need to be executed in real-time. For instance, in a catalog, finding the product whose type is “digital camera” and whose price has decreased.

As mentioned above, the deltas, that we consider here, are XML documents summarizing the changes. The choice of XML is motivated by the need to exchange, store and query these changes. XML allows supporting better quality services as in Chen et al. (2000) and Nguyen et al. (2001), in particular query languages (www.w3.org/TR/xquery), Aguilera et al. (2000), and facilitates data integration (www.w3.org/rdf). Since XML is a flexible format, there are different possible ways of representing the changes on XML and semi-structured data Chawathe et al. (1998), La Fontaine (2001), Marian et al. (2001), XML Update Language (xmldb-org.sourceforge.net/xupdate), and build version management architectures Chien et al. (2001). In Section 3, we compare change representation models and we focus on recent proposals that have a formal definition, a framework to query changes and an available implementation, namely DeltaXML La Fontaine (2001), XyDelta Marian et al. (2001), XUpdate (xmldb-org.sourceforge.net/xupdate) and Dommitt (www.dommitt.com).

**Change detection.** In some applications (e.g. an XML document editor), the system knows exactly which changes have been made to a document, but in our context, the sequence of changes is unknown. Thus, the most critical component of change control is the `diff` module that detects changes between an old version of a document and the new version. The input of a `diff` program consists in these two documents, and possibly their DTD or XMLSchema. Its output is a `delta` document representing the changes between the two input documents. Important aspects are as follow:

- **Correctness:** We suppose that all diffs are “correct”, in that they find a set of operations that is sufficient to transform the old version into the new version of the XML document. In other words, they miss no changes.
- **Minimality:** In some applications, the focus will be on the minimality of the result (e.g. number of operations, edit cost, file size) generated by the `diff`. This notion is explained in Section 2. Minimality of the result is important to save storage space and network bandwidth. Also, the effectiveness of version management depends both on minimality and on the representation of changes.
- **Semantics of the changes:** This is a challenging issue that can help users to understand “what happens” in the real world represented by the XML data. Some algorithms consider more than the tree structure of XML documents. For instance, they may consider keys (e.g. ID attributes defined in the DTD)