CS4 Dissertation

XML Diff and Patch Utilities

Adrian Mouat

Supervisor: Dr. Joe Wells

June 4, 2002

Declaration

I, Adrian Mouat, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: .....

Date: .....

Abstract

Standard UNIX tools exist for comparing (diff) and patching (patch) files, which operate on a line by line basis using well-studied methods for computing the longest common subsequence (LCS). Using these tools on hierarchically structured data leads to sub-optimal results, as they are incapable of recognizing the tree-based structure of these files. This document introduces a project to create XML diff and patch utilities which operate on the hierarchical structure of XML documents.

Acknowledgements

I would like to thank Dr. Joe Wells for his help and guidance,

CS4 for putting up with my constant whinging,

Mustafa Iqbal for proof-reading the document, and "The Black Rebel Motorcycle Club", whose music kept me (somewhat) sane.

Contents

Chapter 1

Introduction & Background

The aim of this dissertation was to create XML-based equivalents of the UNIX diff and patch tools. The utilities and source code are available on-line at `http://diffxml.sourceforge.net`.

This document is comprised of 8 chapters, covering different aspects related to the design and creation of the tools:

- The rest of this chapter is dedicated to an introduction to XML and tree-differencing, as well as an overview of possible applications.
- Chapter 2 discusses related work, both in terms of tree-differencing algorithms and existing software for comparing XML documents.
- Chapter 3 details the aims and objectives of the project, and the functional requirements.
- Chapter 4 covers the design of the tools and the algorithms implemented.
- Chapter 5 is a detailed specification for a generalised output format for showing the difference between XML documents.
- Chapter 6 reviews the implementation of the utilities; the technology used, how the programs are called and the public release of the utilities.

- Chapter 7 details the testing of the programs; both the process used and the results.
- Chapter 8 reviews the achievements and limitations of the work, and suggests possible extensions.
- The appendix contains a copy of GNU General Public License, under which the tools were released.

1.1 XML

As stated by the World Wide Web Consortium (W3C) [1], "The Extensible Markup Language (XML) is the universal format for structured documents and data on the Web." XML is designed to be interoperable with both SGML and HTML. A section of a possible XML document is shown in figure 1.1.

```
<media type="CD">
        <length>10m15s</length>
        <artist>Exemplar</artist>
        <track num="1">
                <title>Hello</title>
                <length>5m32s</length>
        </track>
        <track num="2">
                <title>Goodbye</title>
                <length>4m43s</length>
        </track>
</media>
```

Figure 1.1: Part of an XML document

XML documents are made up of one or more *elements* which are delimited by *tags*. *Attributes* can be included inside tags to give further information about the element. Elements are ordered whilst attributes are unordered. In the previous example '<length>' is a tag with no attributes whilst '<track num="1">' is a

tag with one attribute. Note that the structure of XML is strictly nested, for example the document in figure 1.2 would be illegal, as the "media" element is terminated inside a child element.

```
<media type="CD">
        <length>10m15s</length>
        <artist>Exemplar</artist>
        <track num="1">
                <title>Hello</title>
                <length>5m32s</length>
        </track>
        <track num="2">
</media>
                <title>Goodbye</title>
                <length>4m43s</length>
        </track>
```

Figure 1.2: Part of an illegal XML document

1.2 Trees and Differencing

The strict nesting of XML allows us to represent XML documents as ordered trees. Our first example in figure 1.1 could be shown as the tree in figure 1.3 (ignoring attributes).

It is clear that the problem of finding the changes between two XML documents can be seen as the "Tree-to-tree Correction Problem" [2] for ordered labeled trees.

Consider the two trees in figures 1.4 and 1.5:

We wish to apply a set of operations to Tree 1 to create Tree 2. The most basic operations we can apply are:

- change the label of a node
- delete a leaf node

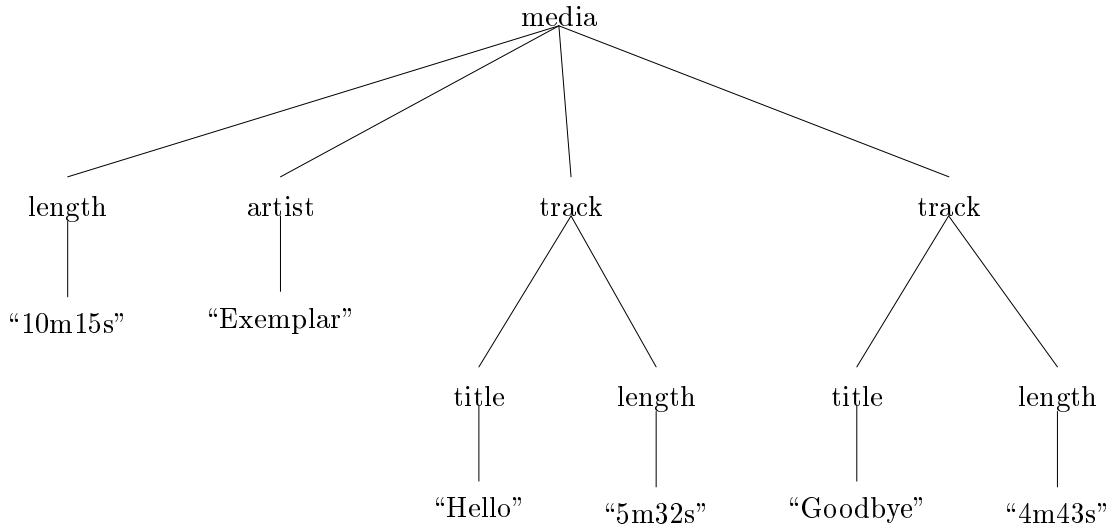

Figure 1.3: Tree representation of an XML document

- insert a leaf node

We will call a set of such operations an *edit script*. The set of edit scripts which transform Tree 1 into Tree 2 is infinite; we could continuously add and delete nodes. However we want to find a *minimal* edit script which transforms Tree 1 into Tree 2. An example edit script to change the tree in figure 1.4 into the tree in figure 1.5 could be:

- delete L(a), the first child of node 2,
- add L(f) as the 1st child of node 3,
- relabel the 2nd child of node 4 from A(e) to A(g).

1.3 Output Format

To be of more value to users, we need to be able to show the edit script in a more intuitive, visual and immediately discernible format. An obvious and flexible method of solving this problem is to change the edit script into a format which is valid XML and can be used by other programs. This format can then be modified, e.g. by an XSLT transformation into a format which displays the changes in a form which is easy to read by users.

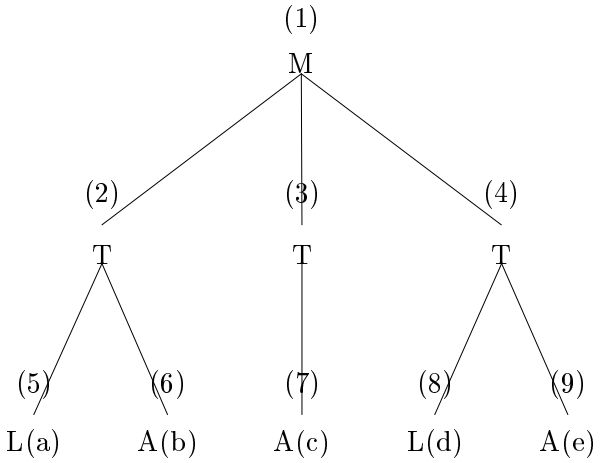

Figure 1.4: Tree 1

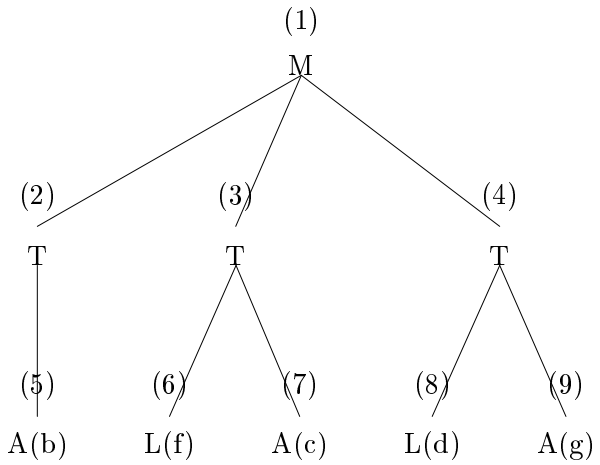

Figure 1.5: Tree 2

We use the term *delta* in the revision control sense to mean a representation of the differences between two objects, in our case XML documents. This sense of the word probably came from its use in mathematics and engineering where it can mean a "quantifiable change".

1.4 Patching

The problem and action of applying a delta to a file in order to produce a new version of the file with the changes incorporated is known as *patching*. The file can be, but is not necessarily, one of the files used in the creation of the delta.

A delta used in this context may be called a *patch*, and the file with the changes incorporated may be referred to as the patched file.

The term *patching* is used to refer to the problem and action of applying such a patch.

1.5 Applications

There are many possible applications for XML differencing and patching tools, some conceivable uses are:

Version Control: Text based version control systems use the standard UNIX diff and patch tools extensively. Version Control systems covering XML would greatly benefit from an XML diff tool. Although the UNIX diff utility will produce valid output for XML files, the output will be sub-optimal in comparison to a diff utility cognizant of the hierarchical structure of the data. The hierarchical delta should also capture the "essence" of any changes - what the users intentions were when modifying the file - much better than the line-oriented diff.

Document Comparison and Updating: XML documents written by an author or co-author can be checked to find the changes between versions. "Patches" can be distributed containing the changes made by an author, and others can choose whether or not they wish to apply the changes to their copy of the document.

Databases: XML is increasingly used for storing data in databases. Detecting changes to data is important for many database applications. The XyDiff [11] program was developed specially for the Xyleme [17] data warehousing project. For example if the database returns XML documents for query

results, we can identify the nature of any changes to a standing query e.g. detect when a new name is added to a mailing list.

Web Caching: Currently web caches must request complete documents if they do not hold a current version of the requested page. Using a differencing utility, they need only request a delta between the cached page and the new page. This could create a large reduction in the amount of web traffic, and result in improved transfer times for users. Such a system could cache dynamic as well as static web objects. See [9] for more information.

Transaction Data: If a user has a common query against an application it would be possible to send only a delta of any changes to the previous query result rather than send the complete document again. For example a sports ticker application could send information on only the current event (e.g. a goal being scored, a yellow card given), rather than send the full account of the match to date. This can result in significant bandwidth savings.

Chapter 2

Related Work

This chapter details previous work carried out in areas relevant to the project. First we look at tree-correction algorithms that can be used in differencing XML documents, before moving on to look at existing products for differencing XML and their output formats.

2.1 Tree Correction Algorithms

As explained in the previous chapter, the problem of finding the changes between two XML documents can be seen as the tree-to-tree correction problem. This section covers several algorithms created to solve this problem.

2.1.1 The Extended Zhang and Shasha Algorithm

Barnard, Clarke and Duncan's paper [2] gives a concise overview of early (pre-1995) work on the tree-to-tree correction problem. As the early work has largely been superseded by later algorithms and papers, we will not consider it here. However the paper also proposes an algorithm based on Zhang and Shasha's work [3] which we will refer to as the Extended Zhang and Shasha (EZS[1]) algo-

[1]The name EZS is coined in the *xmldiff* [4] documentation

rithm.

The original algorithm by Zhang and Shasha [3] runs in time $O(n^2 \log^2 n)$ for balanced trees [5], where n is the number of tree leaves (worse for unbalanced trees). The algorithm uses the following *primitives* (basic operations):

- *change* change the "value" of a node to a new value, e.g. replace the text of a sentence
- *delete* a leaf node
- *insert* a leaf node

Barnard, Clark and and Duncan extended Zhang and Shasha's algorithm by adding the following primitives which act on subtrees rather than just nodes:

- *deleteTree* deletes a subtree
- *InsertTree* inserts a subtree
- *swap* swaps a subtree with another subtree

These operations were added to give better edit scripts for documents; they allow operations closer to those a user could be expected to perform, such as merging and moving whole sections of text at a time.

The impact these extensions have on the overall time is relatively negligible compared to the benefits. Note that the EZS algorithm will always produce a edit script that is minimal in terms of the costs of the operations.

This algorithm is implemented in the xmldiff [4] program.

2.1.2 The Fast Match Edit Script Algorithm

Chawathe, Rajaraman, Garcia-Molina and Widom's paper [5] covers the Fast Match Edit Script or FMES[2] algorithm. The FMES algorithm was created after

[2]The name FMES is coined in the *xmldiff* [4] documentation

the EZS algorithm and is intended to be complementary to it.

The FMES algorithm uses the following primitives:

- *Insert* inserts a new leaf node
- *Delete* deletes a leaf node
- *Update* changes the "value" of a node to a new value, e.g. replace the text of a sentence
- *Move* moves a subtree from one parent to another

The algorithm splits the tree-to-tree correction problem into two parts; finding a good matching between trees (Good Matching problem) and finding a Minimum Conforming Edit Script (MCES). A description of the operation of the algorithm can be found in section 4.3.1

In order to achieve good performance from the algorithm, it is assumed that for a leaf l in a document, there exists at most one leaf in the other document which "closely" resembles l. This assumption allows the algorithm to perform efficiently, but in cases where this assumption does not hold it may not produce a minimal edit script.

The FMES algorithm runs in order $O(ne + e^2)$ time where n is the number of tree leaves and e is the "weighted edit distance" (described in the paper). Because of the tradeoffs between performance and minimality of edit scripts, the authors suggest using the EZS algorithm in domains where the amount of data is small and the FMES algorithm in domains where there is a large amount of data.

The FMES algorithm is also implemented in the *xmldiff* [4] program.

2.1.3 The xmdiff Algorithm

The xmdiff algorithm presented in [6] is unique in that it defines an external-memory algorithm which can handle arbitrarily long files. The paper is written by Sudarshan Chawathe, a co-author of [5], and represents some subsequent work he has carried out in the area.

The following primitives are used by xmdiff:

- *Insert* inserts a leaf node
- *Delete* deletes a leaf node
- *Update* changes the "value" of a node to a new value

The algorithm uses the idea of edit graphs to reduce the problem of finding a minimum-cost edit script to the problem of finding a shortest path from one end of the edit graph to the other.

In an external-memory algorithm the overriding performance factor is the number of I/O operations. The algorithm can make use of surplus RAM to reduce I/O cost. Given a block size of S, input trees of size M and N respectively, $m = M/S$ and $n = N/S$, the costs are:

- **I/O** $4mn + 7m + 5n$
- **RAM** $6S$
- **CPU** $O(MN + (M + N)S^{1.5})$

2.1.4 Other Algorithms

There exist many other algorithms and papers on the tree-to-tree correction problem, which, due to lack of space, are not covered in depth here, but two in particular deserve a mention:

- Cole, Hariharan and Indyk's paper [7] is recent and achieves an impressive time bound, but is heavily mathematical and it would take some time to understand well enough to create an implementation based on it.

- Chawathe and Garcia-Molina's paper [8] covers the MH-DIFF algorithm. They include primitives to move and copy entire subtrees, which as discussed in the EZS algorithm, can lead to more appropriate deltas for documents. Their work covers *unordered* trees which are not always applicable to XML documents.

2.2 Existing Products

There are several existing products for finding changes between XML products. All of these tools are designed to take two XML files as input and somehow display the changes between them.

IBM's XML Diff and Merge Tool [20] is not covered as it is not designed to produce standalone delta files. Instead the program highlights the differences within a Java GUI. However, IBM's other product XML TreeDiff [22] is considered.

2.2.1 DeltaXML

DeltaXML [23] is proprietary software created by Monsell EDM Ltd.

Interestingly it can handle both ordered and unordered trees. If a Document Type Definition (DTD) is present it is used to obtain entity expansions and default attribute values. Output is either a delta or the original document with changes tagged. The delta format is considered in section 2.3.

2.2.2 xmldiff

The xmldiff [4] product is GPL-licensed free software created by Logilab as part of the NARVAL project.

The program was written in Python and implements the FMES and EZS algorithms. It has two output formats for deltas, one of which is not in XML format and the other is in the XUpdate [24] language (considered later).

The program needs to hold the XML files in an internal structure in memory, hence it cannot handle very large files. Also there are several cases where the program produces incorrect output, due to coalescing of text nodes in XPath (see the XPath standard [29] for more information).

2.2.3 XML TreeDiff

XML TreeDiff [22] product is proprietary software created by IBM.

The program was written as a set of Java Beans intended to mimic the functionality of the traditional UNIX diff and patch programs. It purportedly achieves good performance by the use of "fuzzy subtree matching". The program has 2 output formats, FUL and XUL, of which we consider XUL later, as XUL is the successor of FUL.

2.2.4 XyDiff

The XyDiff program [11] was developed by the VERSO team for INRIA [12].

The program was developed for the Xyleme [17] XML data warehousing project. The utility uses the Xerces [13] C++ parser. At its heart is a very fast algorithm able to difference large (>10Mb) documents. However the algorithm often produces non-minimal output.

XyDiff was released under the open source Q Public License.

2.2.5 diffmk

The diffmk utility [18] is a Perl program written by Norman Walsh of Sun Microsystems.

Although the source code is available, it does not appear to have an Open Source [32] license and remains the copyright of Sun Microsystems. The program uses a Perl algorithm for computing the Longest Common Subsequence (LCS) of two strings. It does not always produce minimal, or even correct output. The output is the original document with changes marked. Distributed with a utility which displays the differences between the files using colours in a way which is easy to read by humans.

2.2.6 XML Diff and Merge Tool

The XML Diff and Merge Tool [19] is proprietary software created by Dommitt Inc.

There is no downloadable evaluation, only an on-line demonstration which invites the user to upload XML files. It uses the xmdiff [6] algorithm. The output is the original document with changes marked.

2.2.7 VM Tools

The VM Tools [21] package contains XML differencing and patching tools.

The tools are written in Java and have a defined API for integration with other java programs. The package is released under their own VM Systems software license. VM Tools does not support differencing of XML processing instructions or comments, nor does it have support for large files.

2.3 Output Formats

All of these products have separate output formats. In this section we consider and contrast the best of them. I have kept this section separate from the discussion of the products as the output formats can stand independent of their implementations.

None of the output formats produce enough context information to produce accurate patches on files considerably different from those used in creating the delta. More useful context information would be, for example, showing any parent and sibling nodes.

For the sake of clarity the examples given in this section have been indented and formatted; the reader should not expect the programs to produce identical output.

2.3.1 DeltaXML

An example of the *DeltaXML* output format is given in figure 2.1. The program has been used to produce a delta between 2 HTML documents where the only change is that the text of a "<td>" element has been changed from "td 3" to "td 3a".

Delta files produced by DeltaXML always have a namespace for DeltaXML associated with them.

The DeltaXML format conveys change information in a non-complex fashion and precisely. However it does not make good use of XPath [29], and seems to contain a lot of redundant information (the unchanged nodes), yet does not provide the context information that is needed for patching changed files.

Monsell have applied for a patent on the Delta XML output format.

```
<xhtml:html xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
  xmlns: xhtml="http://www.w3.org/1999/xhtml" deltaxml:delta="WFmodify">
<xhtml:html deltaxml:delta="WFmodify" >
<xhtml:head deltaxml:delta="unchanged" > </xhtml:head>
<xhtml:body deltaxml:delta="WFmodify" >
  <xhtml:table deltaxml:delta="WFmodify" >
    <xhtml:tr deltaxml:delta="WFmodify" >
      <xhtml:td deltaxml:delta="unchanged" > </xhtml:td>
      <xhtml:td deltaxml:delta="unchanged" > </xhtml:td>
      <xhtml:td deltaxml:delta="WFmodify" >
        <deltaxml:PCDATAmodify>
          <deltaxml:PCDATAold>
            td 3
          </deltaxml:PCDATAold>
          <deltaxml:PCDATAnew>
            td 3a
          </deltaxml:PCDATAnew>
        </deltaxml:PCDATAmodify>
      </xhtml:td>
    </xhtml:tr>
  </xhtml:table>
</xhtml:body>
</xhtml:html>
```

Figure 2.1: DeltaXML output

XUpdate

The XUpdate [24] format is used by *xmldiff* and has the advantage of being fully specified in a recommendation created by the XML:DB [25] initiative. XUpdate can be shaped to a certain extent by the implementation, but it essentially consists of commands as shown in 2.2. The delta represents adding an element "<town>" with the value "San Francisco" inside an element "<address>", followed by appending another element "<address>" as the last child of "<addresses>".

The recommendation for XUpdate is easy to understand, and makes use of the XPath standard. The fact that there exists a standard for XUpdate enables it to be easily adopted by others.

XUpdate's disadvantages are its verbosity and lack of support for context information for the purpose of patching documents other than those from which

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
 xmlns:xupdate="http://www.xmldb.org/xupdate">
<xupdate:element name="address">
   <town>San Francisco</town>
</xupdate:element>
<xupdate:append select="/addresses" child="last()">
   <xupdate:element name="address">
      <town>San Francisco</town>
   </xupdate:element>
</xupdate:append> </xupdate:modifications>
```

Figure 2.2: XUpdate Output Format

the original delta was computed. Also there is no support for selecting only *part* of a text node, which is useful in creating small deltas.

XUL

The XUL output format is used by the IBM *XML TreeDiff* program. IBM have spent a reasonable amount of time developing XUL, using XUL to replace FUL as the default output format for *XML TreeDiff*.

An example of XUL output is given in figure 2.3.

```
<node id="/*[1]" />
        <node id="/*[1]/*[2]" />
        <node op="add" name="B" type="3" />
                <node id="/*[1]/*[3]"/>
                <node op="add" name="G" type="3" />
        </node>
</node>
```

Figure 2.3: XUL Output Format

An understanding of XPath [29], not covered here, is required to understand this output format.

The format is not very readable as nodes are referred to as numbers, not by their names or values. Although this output format is of limited help to a user, from the machine's point of view it could make for faster and easier patching,

when patching one of the same documents on which the delta was produced. An important point of this format is that the delta itself is in a hierarchical format, which is helpful if we are to add context information.

2.4 Conclusion

From the algorithms covered earlier, the most appropriate algorithms seem to be xmdiff [6] and FMES [5].

The xmdiff algorithm allows differencing of large files and produces minimal edit scripts, both points which are important to creating a useful diff utility.

The FMES algorithm does not always produce minimal deltas and only works in main memory, but should run substantially faster. In many applications it is preferable to quickly see the changes at a glance rather than wait longer and be given a slightly more minimal delta.

From the output formats described earlier the two most apt formats are XUpdate and XUL.

XUpdate gives a more textual account of changes and is to some extent a standard, whilst XUL gives a precise and less verbose account of changes that is more appropriate for programs.

Neither of the output formats support extra context information, which is necessary to produce good patches for documents other than those from which the delta was computed.

Overall, although algorithms exist which are capable of efficiently calculating changes, there is no product which includes all of the following qualities:

- An output format that is good for patching changed files,
- A fast and accurate algorithm,
- The ability to handle large files,

- An open source license,
- Not strongly tied to a particular XML parser,
- Has an independent and fully specified output format.

Chapter 3

Requirements

This chapter covers the aims and objectives of the project and the requirements for the utilities. The requirements are broken into input and output of the programs, functional requirements and non-functional requirements.

3.1 Aims and Objectives

3.1.1 Aim

- Provide GPL-licensed [34] free software implementations of XML oriented diff and patch utilities.

3.1.2 Objectives

- Create an XML "diff" utility which finds and outputs the changes between 2 XML documents.
  - Implement algorithm(s) for solving the tree-to-tree correction problem.
  - Define an output format for displaying a delta of the 2 documents. The output format is intended to be used by other programs and not directly read by humans.

- Create an XML "patch" utility which applies a delta from the diff utility to an arbitrary XML document.

3.2 Input and Output

The various inputs and output to the program are detailed in this section.

3.2.1 Input

Diff Utility

- 2 XML documents to be differenced.
- Command line switches for the various options defined in the Functional Requirements section 3.4.

Patch Utility

- Delta output from the diff utility.
- XML document that the delta is to be applied to.
- Command line switches for the various options defined in the Functional Requirements section 3.4.

3.3 Output

The various program outputs are detailed in this section.

3.3.1 Diff Utility

- A delta of the changes between the documents *or*
- A statement of whether the two documents differ if in "silent mode".

The output format must be concise yet allow for the addition of context data. The addition of context data is important to allow patches of files which were not used in the creation of the delta. This context data must be enough to find an appropriate point in the document to apply each change.

It should be possible to easily change the output into a format more easily read by a user. XSL Transformations (XSLT) [31] could be used to transform a generic XML output format into different formats, more appropriate for other purposes. For example we could have transformations to create untagged ASCII output or formats designed to highlight certain aspects of the delta, such as added nodes.

3.4 Functional Requirements

This section describes requirements which directly affect the functionality of the tool. The requirements are broken into primary and secondary requirements, reflecting their relative importance. Some discussion of why the requirements are necessary is included.

3.4.1 Diff Utility

Primary Requirements

- The tool must be able to read in 2 well-formed XML documents, *file1* and *file2*, and output a set of differences that can be used to create *file2* from *file1*. This is the base functionality required from the program.
- The utilities must take a similar form to the existing UNIX diff and patch tools. This will make the program much more intuitive and usable by UNIX users.
- The program must operate on the tree-structure of the XML files as opposed

to its flat line-based structure. This is necessary to produce deltas which properly embody the meaning of the changes between the files.

- The tool must be able to handle arbitrary length files. A utility which only works on small files is of limited use.
- The ability to add context information to deltas. This will allow accurate patching of documents which are not *file1* or *file2*.

Secondary Requirements

- A choice of algorithms should be available, one which always produces minimal deltas, and one which is faster but may not always produce minimal deltas. In some cases users will want a minimal delta, in other cases they may want to sacrifice minimalness for speed.
- Options to ignore whitespace and character case within nodes. Various options for the stripping of whitespace are possible, e.g;
  - Never strip whitespace.
  - Always strip leading and trailing whitespace.
  - Only strip whitespace if parent element is in a given list.
  - Only keep whitespace if parent element is in a given list.
- An option to ignore changes to XML comments in a delta. Users may not be interested in changes between comments, and may want to turn this functionality off.
- A "silent mode" which outputs only whether or not two files differ. It would be useful to include a simple check to tell if two files differ, that does not output an entire delta.

3.4.2 Patch Utility

Primary Requirements

- The tool must be able to take deltas from the diff tool and apply them to XML documents. In cases where the delta is applied to the same document used as *file1* in the diff, the patch program must produce output equivalent to *file2*. This is the most basic function required of the program.

- The tool should be able to *reverse* the sense of a patch; e.g. change all adds to deletes and vice-versa.

- The tool should be able to apply patches to XML documents other than those used to complete the original delta. Patching such documents may necessarily be less exact and whether or not to apply a particular change will depend on the mode of operation and the accuracy of the match. Discarded changes should be placed in a "reject" file.

- The tool should include controls over its level of "interactivity". "Interactivity" is defined as the ability to query the user on whether or not a change should be applied. The user should be able to specify the level of interactivity, ranging from always query to never query. This option is useful if the user needs to have control over which changes are to be applied.

Secondary Requirements

- Options to ignore whitespace and character case. The user should be able to specify if changes in whitespace and case are unimportant, and should not be applied.

- The tool should mimic the original UNIX "patch" controls which include a "fuzz-factor" which determines when a match is a good one, based on

context. If the match is not good, the change is rejected or the user is consulted. It makes sense to keep this functionality the same if it does not affect usability.

3.5 Non-Functional Requirements

This section covers requirements which do not affect the functionality of the tool, but are nonetheless important.

- The tools will be put under the GPL license, which allows others to freely use and extend the program.
- The tool's usage, output and structure will be clearly documented, to help others who may wish to extend or modify the tool, as well as normal users.
- The tools must run on the departmental Linux machines.

Chapter 4

Design

The utilities were designed using the process oriented method put forth in [27]. This method was chosen due to previous experience with it, and because the structure of the program readily breaks down into a hierarchical, functional flow modelled well by this method.

An object-oriented methodology, such as the UML, was not used mainly because I have little experience of using such a methodology. The program also does not decompose as readily into objects as a functional flow, as it is mainly composed of two large, inalterable algorithms.

4.1 Diagrams and Charts

All diagrams and the data dictionary can be found online at `http://diffxml.sourceforge.net/design/`. There are three types of diagram used:

- **Data Flow Diagrams (DFDs):** Model the logical process of the program. Shows the processes which compose the program and how data is passed between them. For each DFD with a control process, an STD is also included. A data dictionary contains definitions for the dataflows and stores in the DFDs.

- **State Transition Diagrams (STDs):** Records the control information required within the real-time logical process model. Shows the various states a program can pass through.

- **Structure Charts:** Models the system as hierarchical, synchronous, interacting modules. As opposed to a DFD which is asynchronous and has no explicit hierarchy.

4.2 Relating XML documents to trees

Both the FMES and the xmdiff algorithms work on rooted, ordered, labeled trees where each can contain some "value".

A rooted tree has exactly one node that has been selected as the basis of the document, as opposed to a "free tree" or acyclic graph. An ordered tree is where the children of nodes have a designated order. In a labeled tree each node has a name (label) which is not necessarily unique but in some sense defines its "type", for example a sentence or paragraph in a document tree. By the value of a node we mean whatever information it holds, for example the contents of a sentence in a document tree.

It is important that we relate these concepts to XML files, so as to remove any ambiguity. To do so we will reference definitions given in the Document Object Model (DOM) Level 2 Core Specification [10].

Firstly, XML can be seen in a hierarchical format because XML is *strictly nested*; XML elements must always be properly closed and may not overlap. XML files can always be seen as rooted trees as there is always exactly one root element (the "document element" in DOM) corresponding to the root of the tree. The labels of an XML document are the values returned by the DOM *getNodeName()* method. The exception to this rule is attributes, which, in our

context, are not defined as nodes but as the "value" of an element. The value of nodes, other than attributes and elements are the same as that returned by the DOM *getNodeValue()* method.

The breakdown of value and label for each considered node type is:

Node Type	**Label**	**Value**
Element	tag name	((attribute title)(attribute value))*
Comment	#comment	content of the comment
Text	#text	content of text node
Processing Instruction	target	entire content excluding the target

Several node types are not considered, either as they are only applicable to DTDs[1], or are not considered leaf nodes. The value of Element nodes, "((attribute title)(attribute value))*", represents an associative array of attribute titles with their values.

Using these definitions it is possible to build a tree of the form usable by the FMES and xmdiff algorithms from an XML file.

4.3 Description of Algorithms

The following section briefly describes the working of the FMES and xmdiff algorithms. The algorithms build upon or make use of similar concepts which are described first. Both algorithms create *Edit Scripts*, a sequence of edit operations which transform one tree into another. We use the definition of edit operations as described in [5].

We consider four main edit operations:

[1]Document Type Definition, see the XML specification [1]

- **Insert:** The *insertion* of a new leaf node x into T_1, denoted by INS((*x*,*l*,*v*),*y*,*k*). A node x with label l and value v is inserted as the kth child of node y of T_1. Where x is some unique node identifier. More precisely, if $u_1, \ldots, u_m$ are the children of y in T_1, then $1 \leq k \leq m+1$ and $u_1, \ldots, u_{k-1}, x, u_k, \ldots, u_m$ are the children of y in T_2. The value of v is optional and is assumed to be null if omitted.

- **Delete:** The *deletion* of a leaf node x of T_1, denoted by DEL(x). The result T_2 is the same as T_1, except that it does not contain node x. DEL(x) does not change the relative ordering of the remaining children of *p(x)*. This operation deletes only a leaf node; to delete an interior node we must first move its descendants to their new locations or delete them.

- **Update:** The *update* of the value of a node x in T_1, denoted by UPD(*x*,*val*). T_2 is the same as T_1 except that in T_2, $v(x) = val$, where $v(x)$ denotes the value of a node x.

- **Move:** The *move* of a subtree from one parent to another in T_1, denoted by MOVE(*x*,*y*,*k*). T_2 is the same as T_1, except x becomes the kth child of y. The entire subtree rooted at x is moved along with x. This operation is not supported by the xmdiff algorithm.

In most cases there are many edit scripts that will change T_1 into T_2. We want to choose an edit script which does the minimum amount of work necessary. In order to formalize this notion it is necessary to have a cost model which assigns a cost to each operation. This requires the functions:

- *ci(x)*: which returns a positive number representing the cost of inserting a node x.

- *cd(x)*: which returns a positive number representing the cost of deleting a node x.
- *cm(x)*: which returns a positive number representing the cost of moving a subtree rooted at x.
- $cu(v_1, v_2)$: which returns a positive number representing the cost of updating a value from v_1 to v_2.

The numbers returned should be consistent with regards to each other, for example the cost of updating a node's value to a similar value should be less than the cost of deleting the node and inserting a new node. The cost of an edit script is the sum of the costs of its individual operations.

4.3.1 The Fast Match Edit Script (FMES) Algorithm

The FMES algorithm is fully described in [5].

The algorithm splits the problem of finding the minimum cost edit distance between ordered trees into two subproblems:

- The "good" matching problem.
- The minimum "conforming" edit script problem.

The "good" matching problem is finding an appropriate matching between the nodes of two trees, T_1 and T_2, that can be used in solving the minimum "conforming" edit script problem. Two nodes are said to have a matching if the nodes have similar or identical values. Matchings exist on a one-to-one basis. A set of matchings M can be considered better than a set of matchings M' if using M to compute the edit script results in a cheaper edit script than using M'.

For reasons of efficiency, the algorithm assumes that for any given leaf node $y \in T_2$, there is at most one node $x \in T_1$ which is computed to match y. This

assumption will not hold for all documents. In such cases the algorithm may generate non-minimal output, in these cases we trade minimality for speed.

The matching algorithm works by traversing T_1 bottom-up, looking for matches with so far unmatched nodes in T_2, which are added to M. The nodes are then marked as "matched". This basic algorithm can be improved by creating "chains" of nodes with the same label and using Longest Common Subsequence (LCS) [28] algorithms to get an initial matching between nodes.

The minimum "conforming" edit script problem is to create a minimum cost edit script conforming to a set of matchings M, given the set M and two trees, T_1 and T_2, which transforms T_1 into T_2. There are five main stages in the algorithm used to compute the edit script, E. In the following description of the stages, $p(x)$, $l(x)$, $v(x)$ denote the parent, label and value of a node x respectively.

The five stages are:

- **Update:** Look for matched pairs of nodes (formally $(x, y) \in M$) which have differing values ($v(x) \neq v(y)$). For each pair append the edit operation UPD$(x, v(y))$ to E and apply the update to T_1.

- **Align:** The children of a matched pair ($(x, y) \in M$) are *misaligned* if x has matched children u and v such that u is to the left of v in T_1 but the partner of u is to the right of the partner of v in T_2. Each pair of internal matched nodes are checked to see if their children are misaligned. Misaligned children are aligned via a move operation which is then appended to E. For details on how the move operations are worked out, refer to the paper [5].

- **Insert:** Look for an unmatched node $z \in T_2$ such that its parent is matched. Suppose $y = p(z)$, and y's partner in T_1 is x. For each node append edit operation INS$((w, l(z), v(z)), x, k)$ to E, and apply the operation to T_1. Add (w, z) as a matched pair to M. Variable w denotes a new unique

node identifier created for the node, and position k is determined with respect to the children of x and z that have already been aligned. The node inserted becomes a leaf node, any children of z will be inserted as a separate operation.

- **Move:** Look for pairs of matched nodes ($(x, y) \in M$) whose parents are not matched. Append edit operation MOV(x, u, k) to E and apply the operation to T_1. Variable u denotes the matched node of the parent of y in T_1. The position k is determined with respect to the already aligned children, as in the insert phase. Both the parents are added to the matching set M.

- **Delete:** Look for unmatched leaf nodes x in T_1. For each such node add DEL(x) to E and apply the delete operation to T_1.

Once the algorithm has completed, T_1 has been transformed into a copy of T_2, E is the final edit script and M is a matching between all nodes in the trees to which E conforms.

4.3.2 xmdiff

The xmdiff algorithm [6] reduces the tree-to-tree correction problem to the problem of finding a shortest path in the *edit graph* of the two trees.

Edit graphs are used in several algorithms, notably the Myers LCS algorithm [28]. For a full description of edit graphs refer to the Myers paper or the xmdiff paper.

Intuitively an edit graph can be thought of as a simple grid, with the sequences of nodes being compared as its axes. Suppose the sequence of nodes representing T_1 are on the horizontal axis, and the sequence of nodes representing T_2 are on the vertical axis. Therefore each point on the grid has a corresponding node in

T_1 and in T_2. There are directed edges between each node to the node (if any) to the right, bottom and bottom-right. Naturally horizontal edges are directed to the right, vertical edges to the bottom and diagonal edges to the bottom right. Crossing an edge horizontally represents deleting the corresponding node of T_1, crossing an edge vertically represents inserting the corresponding node of T_2 and crossing an edge diagonally represents updating the value of the corresponding node on T_1 to the value of the corresponding node on T_2. Weights are attached to the edges equal to the cost of the edit operations they represent. Therefore any minimum cost edit script will map to a path in the edit graph from the top-left to the bottom-right.

The xmdiff algorithm can be broken into 2 components, computing the *distance matrix* and generating the edit script.

The distance matrix is a $(M+1) \times (N+1)$ matrix D, where M and N are the number of nodes in the respective input trees. $D(x, y)$ is the length of the shortest path from $(0, 0)$ to (x, y) in the edit graph. The computation of the distance matrix is by a simple algorithm which checks the weights assigned to edges in the edit graph.

Generating the edit script is the relatively easy task of following the minimum cost path through the graph matrix and outputting the appropriate edit operation at each step.

This algorithm is extended to compute the differences in external memory by several techniques based in buffering and computing nested-loop joins in relational databases. These techniques are not covered here; for a full account consult the xmdiff paper [6].

Chapter 5

Output Format

This chapter covers the output format options supported by the program. A breakdown and motivation for each of the options is provided. Extensive use is made of the XPath standard[29] and the DOM Level 2 Core Specification [10], which the reader may wish to consult.

5.1 Delta Update Language (DUL)

The natural output of the algorithms is an Edit Script as previously defined. We want our output format to be a well-formed XML document, so that it can be easily used by other programs and modified into other forms, possibly by XSL transformations. The basic XML elements in DUL are defined to be roughly equivalent to the relevant edit script operations. The DUL attempts to model the basic edit script operations as XML elements.

5.1.1 Insert

Syntax

<insert
parent="*xpathexpr*"
childno="*cn*"
charpos="*char*"
nodetype="*code*"
name="*title*"
>*value*</insert>

Description

Inserts a leaf node into the document. The instruction is intended to be equivalent to the edit script operation INS($(x,l,v),y,k$) described on page 31.

Attributes

parent:

The variable *xpathexpr* is an XPath expression that uniquely identifies the parent element, equivalent to y in the edit script operation. The XPath expression is restricted to having node tests of the form "node()", which matches any XPath node, followed by an abbreviated position predicate of the form $[x]$ where x is the position number of the node. The *xpathexpr* uniquely identifies the parent node.

childno:

The variable *cn* is the child number of y that the node is to be inserted as (the old node at this index becomes the *cn+1* node). The number represents the XPath "node()" position taken as child of the parent node (as opposed to the DOM node index). The child number is unused and may be omitted in cases where an attribute is inserted, as attributes have no defined order. The variable *cn* is equivalent to k in the edit script operation.

charpos:

In cases where inserts are made in the middle, immediately after or immedi-

ately before character data, it is necessary to hold the character position at which to insert the node. The variable *char* is the numeric character position at which to insert the node. The first character of a text node is 1, in accordance with the XPath standard. Setting the attribute to 1 is equivalent to inserting before the text. If omitted, *char* defaults to 1.

nodetype:

The variable *code* is the DOM code of the node returned by the DOM *getNodeType()* method, and is part of $l(x)$ in the edit script operation. These codes are given in figure 5.1:

Node Name	**Code**
Element	1
Attribute	2
Text	3
Processing Instruction	7
Comment	8

Figure 5.1: Table of DOM codes

Document nodes, Document Type nodes and Document Fragment nodes (as defined in the DOM Level 2 Core Specification) are not included, as they are not appropriate leaf nodes. As DTDs are not considered within the scope of DUL, we also do not include Notation nodes, Entity nodes, or Entity Reference Nodes. CDATA Sections are seen as Text nodes to avoid problems when using XPath, which does not differentiate between CDATA Sections and other text. The difference algorithm considers attributes only as the value of their parent nodes, but to preserve generality they are considered nodes distinct from their associated elements in DUL.

name:

The "name" attribute is used in cases where an attribute, element or process-

ing instruction is being inserted. The variable *title* gives the name of an attribute, the tag name of an element, or the target of a processing instruction. In cases where the node is not one of these types, it may be omitted. Default is the empty string.

Content

The content of an insert element, *value*, is the DOM value of the node to be inserted, as returned by the DOM *getNodeValue()* method. Equivalent to v in the edit script operation.

The values are given in figure 5.2:

Node Name	Value
Attribute	value of attribute
Comment	content of comment
Element	null
Processing Instruction	entire content excluding target
Text	content of text node

Figure 5.2: Table of DOM Node Values

In cases where the value is defined to be null, including node content has no effect. In these cases the insert operation may be represented by an empty element. Representing cases which do not have a null value by an empty tag is equivalent to setting the value to the empty string.

Example

The implementation of the differencing algorithm does not attempt to match attribute nodes by themselves, instead matching elements whose tag names and attributes match. Therefore one edit script operation to insert an element may be represented by several insert elements e.g:

Inserts the element:

“<section title='Poetry' />”

```
<insert parent="/node()[1]/node()[3]" childno="2" nodetype="1"
name="section" />
<insert parent="/node()[1]/node()[3]/node()[2]" nodetype="2"
name="title" />Poetry</insert>
```

into the document.

5.1.2 Delete

Syntax

```
<delete
    node="xpathexpr"
    charpos="char"
    length="len / >
```

Description

Deletes a leaf node from the document. Elements with attributes but no child nodes are considered leaf nodes for this purpose, and hence can be removed by this operation. The instruction is intended to be equivalent to the edit script operation DEL(x) described on page 31.

Attributes

node:

The variable *xpathexpr* is an XPath expression which uniquely identifies the XPath node to be deleted. Attributes may be deleted by an appropriate *xpathexpr*, which specifies their title. The variable *xpathexpr* is subject to the same restrictions as for an insert, with the exception that when an attribute is being deleted it is specified as the last predicate of the *xpathexpr*.

charpos:

In cases where character data is being deleted, it is necessary to specify how much of the text to delete. The attribute "charpos" is used in conjunction with the "length" attribute to unambiguously specify what text to remove. The vari-

able *char* is the index of the first character to delete, counting in the same way as for the insert operation. Unused in cases where the node is not a text node. If omitted it defaults to 1.

length:

This attribute is used whenever a text node is being deleted. The variable *len* identifies the number of characters to delete, from and including the character specified by the "charpos" attribute. If omitted defaults to 0. Hence if the "length" attribute is unspecified when deleting a text node, no deletion takes place.

Examples

Deleting an attribute:

<delete node="/node()[1]/node()[2]/node()[3]/@title />

Removes the "title" attribute of an element.

An example of deleting a text node is:

<delete node="/node()[1]/node()[4]" charpos="1" length="7" />

Deletes the first 7 characters from the text node identified.

5.1.3 Update

Syntax

<update
node="*xpathexpr*"
charpos="*char*"
length="*length*"
/ >*value*</update>

Description

Updates the value associated with a node. The instruction is intended to be equivalent to the edit script operation UPD(x,*val*) described on page 31.

Attributes

node:

The variable *xpathexpr* uniquely identifies the node to be updated, equivalent to x in the edit script operation. The XPath expression is restricted as for the delete element, with the addition that elements may not be identified. This is because elements have no "value" to update. This is in accordance with the DOM specification where the *getNodeValue()* method returns null for elements. The names of elements and attributes may not be updated.

charpos:

In cases where character data is being updated, it is necessary to specify how much of the text to change. The attribute *charpos* is used in conjunction with the "length" attribute to unambiguously specify which text to update. The variable *char* is the first character to change, counting in the same way as for the insert operation. It is unused in cases where the node identified by *xpathexpr* is not a text node. If omitted it defaults to 1.

length:

This attribute is used whenever a text node is being updated. The variable *len* identifies the number of characters to update, from and including the character specified by the "charpos" attribute. If omitted defaults to 0. The number of characters specified by the "length" attribute are always changed, if the new text is not *len* characters long, the old text is truncated. Similarly if the new text is more than *len* characters, the extra text is inserted without overwriting. Hence if the "length" attribute is unspecified when updating a text node, the new text is inserted at the appropriate position, without overwriting the old text.

Content

The *value* variable represents the new value for the node. The meaning of the value is the same as for the insert operation. In cases where character data is

being updated, the new text overwrites existing characters beginning at *char*. Any characters not overwritten are kept in the original position. Any characters left in *value* after overwriting the original final character are appended.

Examples

An example of updating a non-attribute node is:

<update node="/node()[1]/node()[2]/node()[3]">this is a comment</update>

An example of updating an attribute is:

<update node="/node()[1]/node()[3]/node()[2]/@title" >Arch Bishop</update>

Which changes the value of the "title" attribute to "Arch Bishop".

5.1.4 Move

Syntax

<update
node="*xpathexpr*"
old_charpos="*ochar*"
length="*len*"
parent="*parxpathexpr*"
childno="*cn*"
new_charpos="*nchar* />

Description

Moves the position of a subtree or leaf node within a document. The instruction is intended to be equivalent to edit script operation MOV(x,y,k) described on page 31.

Attributes

node:

The variable *xpathexpr* uniquely identifies the node or subtree to be moved.

The XPath expression is restricted as for the delete element, except that attributes may not be moved.

old_charpos:

In cases where character data is being moved, it is necessary to specify how much of the text to move. The attribute "old_charpos" is used in conjunction with the "length" attribute to unambiguously specify what text is to be moved. The variable *ochar* is the index of the first character to move, counting in same way as for the insert operation. Unused in cases where the node is not a text node. If omitted it defaults to 1.

length:

This attribute is used whenever a text node is being deleted. The variable *len* identifies the number of characters that are to be moved. If omitted defaults to 0. Hence if the "length" attribute is unspecified when moving a text node, no move takes place.

parent:

The variable *parxpathexpr* uniquely identifies the element the node identified by *xpathexpr* is to become a child of. The XPath expression is restricted as for the insert element.

childno:

The variable *cn* is the child number of *parxpathexpr* that the node is to be inserted as (the old node at this index becomes the *cn+1* node). The number is the XPath "node()" position that the node will have (as opposed to the DOM node index). In cases where an attribute is inserted the child number is unused and may be omitted, as attributes have no defined order. Any node currently at position *childno* under *parxpathexpr* is moved to position *childno*+1.

new_charpos:

In cases where moves insert in the middle, immediately after or immediately

before character data, it is necessary to hold the character position at which to insert the node. The variable *nchar* is the numeric character position at which to insert the node, counting in the same way as for the insert operation. The first character of a text node is 1, in accordance with the XPath standard. Setting the attribute to 1 is equivalent to inserting before the text. If omitted, *char* defaults to 1.

Example

An example of a move operation is:

```
<move node="/node()[1]/node()[3]/node()[2]"
        parent="/node()[1]/node()[2]" childno="2">
```

which moves the subtree rooted at the 2nd child of the 3rd child of the root element to be the 2nd child of the 2nd child of the root element.

5.1.5 DUL Example

An example of a complete DUL document is given in figure 5.3.

```
<?xml version="1.0"?>
<DUL>
    <insert parent="/node()[1]/node()[3]" childno="2" charpos="7"
    nodetype="1" name="section" />
    <insert parent="/node()[1]/node()[3]/*[2]" nodetype="2"
    name="title" />Poetry</insert>
    <delete node="/node()[1]/node()[2]/node()[2]" charpos="3"
    length="" 7"/>
    <update node="/node()[1]/node()[3]/node()[2][@title]"
    >Arch Bishop</update>
    <move node="/node()[1]/node()[3]/node()[2]"
    parent="/node()[1]/node()[2]" childno="2">
</DUL>
```

Figure 5.3: Complete DUL Document

The order of internal elements is important as changes are processed with

respect to any previous changes. Note that it is not invalid to do operations on previously modified or added nodes, even pointless cases like inserting then immediately deleting the same element, although it may well be sub-optimal.

This simple representation holds all that is necessary for a delta. In many cases this format will be sufficient. Its advantages are that it is simple and small.

5.1.6 Namespaces

Namespaces [30] are used in XML to qualify element and attribute names by associating them with namespaces identified by Uniform Resource Identifier (URI) references. A namespace should be both *unique* and *persistent*.

When using context information in DUL, it is necessary to use namespaces to differentiate between DUL elements and elements from the documents used to create the delta. It is also possible that a user may wish to use a DUL document or part of a DUL document within another XML document.

DUL's namespace is currently identified as `http://diffxml.sourceforge.net/DUL`. This meets the uniqueness characteristic for a namespace, but, as the Internet host for the project may change, may not meet the persistence characteristic. It was felt that the trade-off was worthwhile in order to provide a URI which contained information on DUL.

5.1.7 Entity References

Currently DUL has no support for showing differences between entity references.

Because of this the current implementation either always resolves entities, or removes external entities from the document. The attribute "resolve-entities" is attached to the root element and is given the value "true" or "false" depending if entities are always resolved or removed respectively. Neither behaviour is entirely correct. Always resolving entities could lead to problems when dealing with

external entities with different URIs; although they may resolve to a certain value at the minute, this value could change at any time. Also it is unclear what should happen in the case that an external server cannot be reached.

Removing entities is also incorrect, as we may be ignoring differences between the documents.

A better solution would involve comparing the URIs of external entities, rather than the values to which the URIs correspond, and extending the DUL to be able to show any differences. This was not implemented as the problem was not realised until late on in the project, and because DTD processing is not considered within the scope of the dissertation.

5.1.8 Adding Context Information

The major disadvantage of the previous output is that it contains very little context information. When we want to apply deltas to documents other than the original, context information helps us to accurately identify which nodes the changes apply to. This reflects the line-oriented diff and patch utilities case where extra lines of context information can be output.

The problem in our case is to decide what should constitute context information.

Although I have not implemented context aware patching, it is worth discussing how context information could be output here.

The following defines several ways of adding context information to DUL documents.

Tag Name Expansion

One of the simplest meaningful additions is element names in place of the "node()" node tests[1] which match any node. For example:

```
<delete node="/doc[1]/chapter[3]/section[2]" />
<move node="/doc[1]/chapter[3]/section[2]"
      parent="/doc[1]/chapter[2]" childno="2"/>
```

Note that the position predicates[2] in the XPath expressions now identify the position with regards to the element name rather than the absolute node position, e.g. in the delete operation we are now talking about the second section element of the third chapter element as opposed to the second child node of the third child node of the root element, which may or may not refer to the same node. Although it would be possible to choose which node tests to expand into names, (e.g. only expand the final step) this level of control is not considered immediately useful enough to warrant the extra syntax and complexity required to include it.

This simple addition makes the meaning of the operation much clearer, but there is still much more that can be added in terms of useful context information.

Reverse Patching

A common operation when patching using traditional line based deltas is to "reverse" the sense of the delta, i.e. inserted lines are deleted and vice versa. This allows the user to recreate the original document in a diff given a patch and the resultant document. In order to reverse the sense of a DUL document, it is necessary to store more information about deleted nodes and updated nodes.

---

[1] Node tests specify the node type and expanded-name of the nodes selected by the location step, see XPath[29] for a full description

[2] Predicates use expressions to refine the set of nodes selected by the location step. See XPath[29] for a full description

To signify that a delta is in a suitable format for reverse patching, the attribute "reverse-patch" with the value "true" should be added to the root element. This attribute defaults to "false" if omitted.

In order to reverse a delete, we need to know the values returned by the DOM methods *getNodeName()*, *getNodeType()*, and *getNodeValue()* for the node to be deleted.

Hence the syntax for the delete instruction becomes:

<delete
node="*xpathexpr*"
charpos="*char*"
length="*len*"
nodetype="*code*"
name="*title*"
/ >*value*</delete>

Where the extra attributes are:

nodetype:

Which is defined as for the "nodetype" attribute for the insert operation. The variable *code* is the DOM code of the node returned by the *getNodeType* method.

name:

Which is defined as for the "name" attribute for the insert operation. The variable *title* is the DOM code returned by the *getNodeName()* method. Only used in cases where the node to be deleted of type element, attribute or processing instruction, and may be omitted in other cases. The default is the empty string.

The content of the delete element, *value*, is the DOM value of the node as returned by the *getNodeValue()* method. In cases where character data is being deleted only the removed characters are included, not the entire contents of the XPath text node. In cases where the value of the node is null, including node content has no effect. In such cases the delete operation may be represented by

an empty element. This is the same as the contents for the insert operation.

In order to reverse an update operation, we need to know the values returned by the DOM methods *getNodeValue()* and *getNodeName()* for the node to be updated. Hence the syntax for the update operation becomes:

<update
node="*xpathexpr*"
charpos="*char*"
name="*title*"
/ >*value*<old>*oldvalue*</old>
</update>

Where the extra attribute is:

name:

Which is defined as for the "name" attribute for the insert operation. The variable *title* is the DOM code returned by the *getNodeName()* method. Only used in cases where the node to be deleted of type element, attribute or processing instruction, and may be omitted in other cases. The default is the empty string.

The character data content of the update element *value* is the new DOM value of the node as returned by the *getNodeValue()* method. The character data of the "old" child element is the original DOM value of the node as returned by the *getNodeValue()* method. In both cases, if character data is being updated only the changed characters are included, not the entire contents of the XPath text node.

Context Nodes

We now consider context information more akin to the flat text concept of outputting context lines surrounding the changed line. In the hierarchical world, we want to be able to not only output the value of sibling nodes but also parent/child nodes and their siblings. In order to provide context information we

need to add extra attributes and elements containing both the context data and information on the context data. Each DUL operation occurs within a context element, which also contains any context nodes. Operations may also contain elements which hold context information. What constitutes a context node is specified by the attributes of the "dul" root element.

An example of a DUL document with context information is shown in 5.4. The example is hard to read due to the absence of whitespace and indentation, which is left out to avoid confusion in identifying context nodes.

```
<?xml version="1.0"?>
<dul:DUL xmlns:dul="http://diffxml.sourceforge.net" sib_context="1"
   par_context="1" par_sib_context="1">
<dul:context
   >text<section><a/><dul:delete
   node="/doc[1]/chapter[2]/section[3]/text()[1]" charpos="1" length="3"
   >234</delete><!- comment -></section>more text<
/dul:context>
</dul:context
   >more text<chapter title="Bits and Bobs"
   ><dul:insert parent="/doc[1]/chapter[3]" childno="1" nodetype="1"
   name="section" /><data/></chapter>even more text<
/dul:context>
<dul:context
   >text<section></a><dul:update
   node="/doc[1]/chapter[2]/section[3]/comment()[1]"
   > another comment <dul:old> comment </dul:old></update
   >description of section</section>more text<
/dul:context>
<dul:context
   >text sibling<chapter><dul:move
   node="/doc[1]/chapter[5]/section[7]" parent="/doc[1]/chapter[4]"
   childno="1" ><dul:context>some text<chapter><sibling/><
   dul:mark><section>some <I> child </I> nodes </dul:mark
   ><sibling/></chapter>end text</dul:context></dul:move>
</dul:context>
</dul:DUL>
```

Figure 5.4: DUL document with context information

Namespaces are used to differentiate DUL elements from context information elements. The attributes attached to the "dul" root element set the parameters

for the context information:

- The attribute "sib_context" sets the number of sibling elements to output around the referenced element. Sibling context is symmetrical; when possible the given number of siblings is output both before and after the given element. The "sib_context" attribute defaults to the value 0.

- The attribute "par_context" sets the number of parent *and child* elements to output around the referenced node. Again context is symmetrical but in all cases except the move operation (and the extended operations considered later), the element will have no child elements. The "par_context" attribute defaults to 0.

- The attribute "par_sib_context" sets the number of sibling nodes to output around the parent/child elements. Such siblings have their values given but not any child elements. The "par_sib_context" attribute defaults to 0. The attribute has no effect when "par_context" is set to 0.

Attributes are not considered to be context nodes within their own right, but are output as part of elements which are context nodes.

The example starts by deleting the text node "234" from the document. Note there are 1 preceding sibling, 1 following sibling, the parent element and the parents preceding and following siblings shown as context. The example then inserts the element "<section/>" into the document. The element has no preceding sibling, so none is shown. This is followed by the updating of a comment node. The format for the update is the same as for the reverse patching format described previously. The final operation is to move a "section" element from "chapter" parent to another. This requires the addition of an "context" child element to the move operation in order to hold the original context of the node. The subtree being inserted is highlighted by enclosure within a "mark" element.

The context for the "insert" and "update" elements refer to the transformed tree, i.e. the context of the node after the change has been applied. Conversely the context for the "delete" element refers to the original tree i.e. the context of the node before the change is applied. The surrounding context for a "move" element refers to the transformed tree i.e. the *new* position of the moved node. The context within a "move" element refers to the original tree i.e. the *old* position of the node being moved. Care needs to be taken not to get false impressions from context information. Note that the context information supplied is fragmented, for example there may be more children associated with parent elements.

5.1.9 Extensions to DUL

It is possible to extend the operations in DUL by considering further operations on subtrees rather than single elements. Rather than unnecessarily define new operations, we overload the meaning of the insert and delete operations:

- **Delete Subtree**

  Delete the subtree rooted at given element:

  <delete node="*xpathexpr*" />

Where the XPath expression *xpathexpr* specifies the root of the subtree to remove. The XPath expression is restricted in the same way as the insert operation. The *xpathexpr* uniquely identifies the element at the root of the subtree being removed. The delete subtree operation is contrasted from the delete node operation as it identifies a non-leaf node.

An example of the delete subtree operation is:

<delete node="/node()[1]/node()[2]/node()[2]" />

This example removes the second child of the second child of the root node and all children below it. Context elements take a similar form to the "delete"

element, but contain a subtree rather than a single node. As text nodes cannot be specified by the *xpathexpr*, we do not need "charpos" and "length" attributes.

- **Insert Subtree**

  Insert a subtree as the *cn*th child element of given parent:

  <insert parent="*xpathexpr*" childno="*cn*" >*subtree*</insert>

Where the XPath expression *xpathexpr* specifies the element to be parent of the subtree. The XPath expression is restricted in the same way as the original insert operation. The *xpathexpr* uniquely identifies the parent element. The subtree is inserted as the *cn*th child of the parent. Any node previously at position *cn* moves to position *cn + 1*. The *subtree* must be well-formed XML, with exactly one root element. The insert subtree operation is contrasted from the insert node operation as the content of the element is a subtree, as opposed to the value of a single node. An example of the insert subtree operation is:

```
<insert parent="/node()[1]/node()[3]" childno="2">
        <section title="tools"><bold><italic>Top Tips</italic>
</bold></section>
</insert>
```

Which appends the given element and its children to become the second child of the given parent. Context elements take a similar form to the original insert operation, but note that all child nodes are shown already in order to perform the insertion.

These elements do not represent the basic operations performed by the differencing algorithms. Instead they build upon the existing operations. In the standard output of the program, the insertion of a subtree has to be represented by several "insert" elements, one for each node in the operation. Similarly the deletion of a subtree has to be represented by several "delete" elements. It should

be possible to replace multiple insertions and deletions of leaf nodes with the subtree operations either by modifying the algorithms or post-processing the delta file.

5.1.10 XUpdate

It was originally intended that the difference program would also support the XUpdate output format. XUpdate is a specification created by the XML:DB Initiative [25]. The current specification document is rather ambiguous, and relies on a reference implementation called Lexus [26], which is devoid of any supporting documentation. As XUpdate only allows a user to specify a XPath text node (which coalesces adjacent text nodes), and not part of text node, we ran into difficulties when trying to delete single DOM text nodes. There are cases when more than a single DOM text node is referenced by a constrained XPath expression. It should be possible to provide work-arounds for these cases but at the time of writing this had not been investigated.

Hopefully XUpdate will be subject to continued improvement, and may gain more widespread usage. For a full description of XUpdate see [24].

Contrast with DUL

The definition for DUL is more rigorous and restricted than that of XUpdate. There is no move operation in XUpdate; it has to be modelled by a delete followed by an insert.

5.2 Human Readable Output

Neither the DUL or XUpdate output format are of a format readable by humans without further processing. As both the formats create well-formed XML documents, we can perform XSL Transformations [31] on the documents to create

output more suited to human consumption. For example by processing the original documents with the delta, it is possible to create an HTML file with changes marked in different colours, e.g. deleted text shown in red, inserted in blue etc.

Chapter 6

Implementation

This chapter covers details of the implementation of the utilities. A breakdown of the technology used in implementing the algorithms is given, as well as a section on invoking the program. Details on the public release of the utilities are also given.

6.1 Technology

In order to aid programming with XML and improve portability, it was decided to make use of the standard Application Program Interfaces (APIs) available for XML. The DOM Level 2 API is used in the implementation of the FMES algorithm and the patch utility. DOM was chosen as it allows XML files to be easily and accurately represented as trees as well as providing easy traversal methods between nodes and their relations.

Certain parts of the DOM Level 3 API were also used. Although DOM Level 3 is still a work-in-progress, certain features were of enough gain that its use was justified despite the reliance on an unstable API. For example the DOM Level 3 methods *getUserData* and *setUserData* were used to avoid having to subclass DOM nodes. The chosen DOM parser is Xerces [13], from the Apache group,

simply because it is a mature and popular parser.

Java was used for almost all the programming. Java was chosen mainly because the DOM APIs are defined in terms of Java methods. Although there are implementations of DOM available for other programming languages, these are all slightly different interpretations of the Java APIs, and tie the utilities to a given implementation. By using Java we therefore increase portability both between platforms and API implementations.

The XML Pull API [15] was used in the implementation of the xmdiff algorithm. DOM was not used as we needed an implementation which avoided storing the document in main memory, in order to be able to process large documents. The Simple API for XML (SAX) could not be used as it has no mechanism for co-ordinating the parsing of multiple documents. The XML Pull API was used as it covers both these issues. The chosen implementation of the XML Pull API was XPP3 [16] a small and extremely fast parser.

XPath [29] is used heavily in our definition of DUL. We used Xalan's [14] XPath API in creation of the patch utility.

All programs used are freely available either under the Apache license (Xerces and Xalan) or the "Indiana University Extreme! Lab Software License" (XPP3).

6.2 Command Line Invocation

This section describes how the tools are invoked from the command line, and the options that can be set.

6.2.1 Diff Tool

Synopsis

diffxml [options] from-file to-file

Finds differences between the XML documents from-file and to-file.

Options

A description of all the options "diffxml" accepts is below. The option names, where sensible to do so, are kept close to those in the GNU diff program. Most options have two equivalent names, one of which is a single letter prefixed with a '-' character, and the other is a long name prefixed by '–'. Multiple single letter options which do not take arguments can be combined into a single command line word, e.g. -vt is equivalent to -v -t.

Long Name	Short Name	Meaning
–brief	-q	Report only if files differ, don't output the delta.
–ignore-all-whitespace	-s	Ignore all whitespace when comparing nodes. Text nodes with only whitespace are not compared.
–ignore-leading-whitespace	-w	Leading and trailing whitespace in text nodes is ignored when comparing nodes. Text nodes with only whitespace are not compared.
–ignore-empty-nodes	-e	Ignore text nodes that contain only whitespace.
–ignore-case	-i	Ignore changes in character case, consider upper and lower case to be equivalent.
–ignore-comments	-r	Ignore changes made to comment elements.
–ignore-processing-instructions	-I	Ignore changes made to processing instruction elements.
–version	-V	Output version number of program.
–help	-h	Print summary of options and exit.
–fmes	-f	Use the FMES algorithm to compute the changes.

–xmdiff	-x	Use the xmdiff algorithm to compute the changes.
–tagnames	-t	Output tag names of elements rather than "node()" for node tests in XPath expressions.
–reverse-patch	-p	Create output with enough information for reversing the sense of a patch.
–remove-entities	-n	Remove all external entities when processing. Allows ignoring of changes to entities and off-line processing, but may produce incorrect results.
–sibling-context [=*nodes*]	-C *nodes*	Create context information output, with *nodes* (an integer) sibling context nodes output to each side of changed nodes. If *nodes* is not given, it will default to 2.
–parent-context [=*nodes*]	-P *nodes*	Create context information output, with *nodes* (an integer) parent and child context nodes output. If *nodes* is not given it will default to 1.
–parent-sibling-context [=*nodes*]	-S *nodes*	Create context information output, with *nodes* (an integer) sibling context nodes of any parent or child context nodes. If *nodes* is not given it will default to 1.

Note that only the DUL output format may have context information.

Exit Status

An exist status of 0 means no differences were found, 1 means some differences were found and 2 means some error occurred.

6.2.2 Patch Tool

Synopsis

patchxml [options] [original file [patchfile]]

Apply an diffxml file to an original.

Options

A description of all the options "patchxml" accepts is below. The option names, where sensible to do so, are kept close to those in the GNU patch program. Most options have two equivalent names, one of which is a single letter prefixed with a '-' character, and the other is a long name prefixed by '–'. Multiple single letter options which do not take arguments can be combined into a single command line word, e.g. -dR is equivalent to -d -R.

Long Name	Short Name	Meaning
–version	-V	Output version number of program.
–help	-h	Print summary of options and exit.
–dry-run	-d	Print results of applying the changes without modifying any files.
–reverse	-R	Assume that the delta file was created with the old and new files swapped. Attempt to reverse sense of change before applying it, e.g. inserts become deletes.

6.3 Public Release

Complied jar files and source code for the diff and patch utilities are available from `http://diffxml.sourceforge.net`. Documentation is also available. A public announcement of the program was posted on `http://freshmeat.net`, as

well as to the comp.text.xml newsgroup. The program is released under the open source GNU General Public License, for which the full license can be found in Appendix A.

I have retained lead developer status on the utilities, and intend to continue their development. By releasing the programs into the environment created by SourceForge [33], I hope to gain the involvement and support of the open source community. Hopefully the programs will see continued improvement, not only by the author but also by other contributors.

Chapter 7

Testing

Testing was focused both on checking that the programs met the requirements in section 3 and on finding any implementation errors or bugs. Testing occurred continuously and in more rigorous explicit testing phases. The following describes the testing that was carried out on the utilities, an example of input and output of the program is available in Appendix B.

7.1 Black Box Testing

Black box testing checks for requirements coverage. The name is derived from the idea that we cannot see the code, only the inputs and outputs, hence the code is a "black" box that we cannot see into. Black box testing therefore centres on finding *faults of omission*, where parts of the specification have not been properly met.

The initial intention was to test the programs with a very large data set. However as cases which caused problems were found relatively quickly, the data set did not become as large as intended. The current data set is available from the project web page at `http://diffxml.sourceforge.net`, containing both correctly handled and incorrectly handled cases.

The following tests were carried out with the data sets:

- Run diff program with all data sets for both output formats.
- Run patch program with deltas and XML documents from above test. Ensure output corresponds to other XML document used in computing delta.
- Run patch program with option to "reverse" sense of patch. Ensure output corresponds to other XML document used in computing delta.
- Run diff program with option to expand tag names in output.
- Run diff program with option to create context output. Values chosen for sibling, parent and parent sibling context should range from 0 to 15.
- Run diff program in "brief" mode for all data sets.
- Run diff program with various options to ignore whitespace.
- Run diff program with options to ignore various elements.
- Run diff program with options to force algorithm used.

A small script which runs a reasonable subset of these tests is available from the project web page `http://diffxml.sourceforge.net`.

7.2 White Box Testing

White box (sometimes known as clear or glass box) testing checks for implementation faults by exercising the boundary values of loops and other parts of the code likely to cause errors. White box testing gets its name from the idea that we can see "into" the code, hence the code is a "white box" we can see into as opposed to a "black box" which we can't.

White box testing occurred continuously throughout development, with values chosen to exercise code paths and boundary values in the module currently being developed. Many bugs were discovered using this method, and nearly all were solved.

7.3 Regression Testing

As the program was developed, a set of working test cases compiled, composing a regression test suite. As changes were made to the program, it was rerun with the old test suite to ensure no previous functionality had been broken. Any new test cases with which the program worked were then added to the test suite.

This ensured that any previously fixed bugs did not reappear in newer versions of the program. The current regression test suite is available from the project web page.

Chapter 8

Discussion

In the final chapter of this document, we discuss the achievements of the project. The initial objectives and requirements are revisited and compared to the outcomes. We also look at possibilities for further development. The chapter concludes with an overview of the project, its achievements and how it sits with previous work.

8.1 Fulfilment of Requirements

In Chapter 3 we defined our objectives as being to:

- Create an XML "diff" utility which finds and outputs the changes between 2 XML documents.
  - Implement algorithm(s) for solving the tree-to-tree correction problem.
  - Define an output format for displaying a delta of the 2 documents. The output format is intended to be used by other programs and not directly read by humans.
- Create an XML "patch" utility which applies a delta from the diff utility to an arbitrary XML document.

As detailed in 6.3, a working implementation of the utilities is available from `http://diffxml.sourceforge.net`. A full definition of DUL, the output format for deltas, is in 5.1. This fully meets all the objectives we laid out with the exception of the "patch" utility which can only apply deltas to the XML documents that were used in creation of said delta.

In terms of the requirements we specified, all of the primary and secondary requirements were met for the diff utility. The requirements for the patch utility to work on XML documents used in creation of the delta and to be able to reverse the sense of patches were met. The other requirements for the patch utility were not fully met. All of the non-functional requirements were met.

Perhaps most important requirement not met was the ability to perform patching on documents other than those used to create the delta. This functionality is a large body of work in itself, and could not be completed due to time constraints. The author believes the original project specification was too ambitious in this respect, and that the timetable underestimated the amount of time needed to fix the bugs. However, the ability of the diff utility to create context output, needed for accurate patching in such cases, was implemented. The patch utility *does* work correctly for cases where the document being patched was used in creation of the delta.

8.2 Limitations and Further Work

This section covers the limitations and possible improvements of the current implementation, as well as functionality additional to the initial requirements. Further details on the limitations are available in 7. These suggestions could take the form of further academic work, or future development of the tools.

- Various efficiency gains are possible.

- Not all pairs of XML documents are handled properly; there are still some bugs in the implementation.
- Although the DUL has support for identifying single character changes, the algorithm still works on a node by node basis; this could be improved.
- Allow input of files over a URL.
- Allow a file name of '-' to stand for standard input and handle properly.
- Creation of API for differencing XML documents. This would require some reworking of existing code, and additional functions, e.g. the ability to difference subtrees only.
- Context patching. Extending the patch utility to be able to apply deltas to documents other than those used to compute the delta. This would require using the DUL context output to match sections of the document with changes to be made.
- Interactive patching. Allow the user to specify if a particular change is to be applied. There should be different levels of interactivity depending on the quality of the patch.
- Options to ignore whitespace and character case in patching. The user could choose whether or not a change is applied if it only affects whitespace or character case.
- Implementation of the DUL extensions described in 5.1.9. These could be implemented by either post-processing the delta file or supporting directly from within the algorithms.
- Work around for XUpdate output format, which has problems when trying to specify character data to delete/insert.

- More output formats. As the deltas produced by the diff utility are well-formed XML, the deltas can be easily transformed into other formats by XSL Transformations [31] or using XML APIs. Formats suitable for particular applications could be created, for example human readable output.
- Improve handling of entity references. This involves some extensions to DUL.
- DUL namespaces properly output by diff utility. Currently they are ignored.
- Support for processing and differencing of XML Schema and DTDs.
- Support for processing and differencing of other hierarchical data formats, e.g. HTML and LaTeX.
- As in some other implementations, e.g. XyDiff, "keys" could be used to force matching of subtrees.

8.3 Conclusion

This document represents the culmination of almost a year of research, design and implementation. The body of work produced is substantial; the source code is in excess of 4500 lines, largely comprised of the sophisticated differencing algorithms, and the DUL specification was no small task in itself.

The project has old roots in both the UNIX diff and patch utilities and the early work on the tree-to-tree correction problem by people such as Zhang and Shasha [3]. The project is also cutting edge, many of the XML standards and software used have only been completed in recent months, some are still at the working draft stage and all are undergoing continued development. A full breakdown of the technology used can be found in section 6.1.

The utilities produced do not represent the only efforts at XML differencing utilities. Existing efforts were covered in the related work section 2. Our work stands out for several reasons, despite its current immaturity:

- It is released under the GNU General Public License, in an environment open to contribution and extensions by others.
- It has its own independent and well defined output language, DUL, as well as support for XUpdate.
- The DUL is unique in that in contains explicit support for context information that can be used in aiding patching of changed files.
- The diff utility supports large documents via the xmdiff algorithm.
- It uses standard APIs to avoid tying itself to particular parsers.

Although the completion of this dissertation marks a milestone in the development of the utilities, it does not mark the end. The author intends to continue development of the programs, and hopefully other members of the open source community or even future dissertations will also build on the current work.

Appendix A

GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and conditions for copying, distribution and modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

(a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

(b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

(c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

(a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as

a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. **Because the Program is licensed free of charge, there is no warranty for the Program, to the extent permitted by applicable law. except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Program is with you. Should the Program prove defective, you assume the cost of all necessary servicing, repair or correction.**

12. **In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the Program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.**

END OF TERMS AND CONDITIONS

Appendix B

Sample Input and Output

The following shows some example input and output of the diffxml utility. The output for the patchxml utility is not shown, as this is simply the original XML file again.

An example of differencing two small XML files:

```
diffxml corr.xml corr2.xml
```

Where corr.xml is the file:

```
<?xml version="1.0"?>
<parent>This element has <child>embedded text</child> within
it.</parent>
```

And corr2.xml is the file:

```
<?xml version="1.0"?>
<parent><!--This element has--><child>embedded text</child>
within it.</parent>
```

Which produces the output:

```
<?xml version="1.0" encoding="UTF-8"?>
<delta><insert charpos="1" childno="1" name="#comment" nodetype="8"
parent="/node()[1]">This element has</insert>
<delete node="/node()[1]/node()[2]"></delete>
</delta>
```

Which is correct (represents inserting the comment and deleting the text).

An example of adding elements and attributes:

```
diffxml attr.xml attr2.xml
```

Where attr.xml is the file:

```
<?xml version="1.0"?>
<parent>This element has <child t="test">embedded text</child>
within it.</parent>
```

And attr2.xml is the file:

```
<?xml version="1.0"?>
<parent>This element has <newchild t="test2">embedded text</
newchild> within it.</parent>
```

Which produces the output:

```
<?xml version="1.0" encoding="UTF-8"?>
<delta><insert charpos="18" childno="2" name="newchild" nodetype="1"
parent="/node()[1]"></insert>
<insert name="t" nodetype="2" parent="/node()[1]/node()[2]">test2</
insert>
<move childno="1" new_charpos="1" node="/node()[1]/node()[3]/node()[1]"
parent="/node()[1]/node()[2]"></move>
<delete charpos="1" node="/node()[1]/node()[3]"></delete>
</delta>
```

Which is correct (shows the insertion the new element and attribute, followed by moving of the text node child).

Larger examples are not given here as the output size quickly grows and reduces in legibility. However, more test cases are available from the website `http://www.diffxml.sourceforge.net`.

Bibliography

[1] World Wide Web Consortium web-pages on XML circa Nov 2001.`http://www.w3.org/XML`

[2] David T. Barnard, Gwen Clarke and Nicholas Duncan. Tree-to-Tree Correction for Document Trees. Queen's University, Ontario, Canada, January 1995.

[3] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing* 18(6):1245-1262, December 1989.

[4] xmldiff by Logilab. `http://www.logilab.org/xmldiff`

[5] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change Detection in Hierarchically Structured Information. Stanford University, California, June 1996.

[6] Sudarshan S. Chawathe. Comparing Hierarchical Data in External Memory. University of Maryland. *Proceedings of the 25th VLDB Conference* pages 90-101, Edinburgh, Scotland, September 1999.

[7] Richard Cole, Ramesh Hariharan and Piotr Indyk. Tree pattern matching and subset matching in deterministic $O(n \log^3 n)$-time. October 2000.

[8] Sudarshan S. Chawathe and Hector Garcia-Molina. Meaningful Change Detection in Structured Data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, p ages 26-37, Tucson, Arizona, May 1997.

[9] Mihut D. Ionescu. xProxy: A Transparent Caching and Delta Transfer System for Web Objects. University of California at Berkeley, December 2000.

[10] The Document Object Model Level 2 Core. World Wide Web Consortium, November 2000. `http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113`

[11] XyDiff by INRIA. `http://www-rocq.inria.fr/~cobena/cdrom/www/xydiff/eng.htm`

[12] INRIA. French national institute for research into data processing and automation. `http://www-rocq.inria.fr/en/campus/index.htm`

[13] Xerces by Apache. XML Parsers in Java and C++. `http://xml.apache.org`

[14] Xalan by Apache. XSLT stylesheet processors, in Java and C++. http://xml.apache.org

[15] Common API for XML Pull Parsing. XML Pull.org. http://www.xmlpull.org/index.shtml

[16] XML Pull Parser 3. An XMLPULL parsing engine. http://www.extreme.indiana.edu/xgws/xsoap/xpp/index.html

[17] Xyleme. http://www.xyleme.com

[18] diffmk by Sun Microsystems. http://www.sun.com/xml/developers/diffmk

[19] XML Diff and Merge Tool by Dommitt Inc. http://www.dommitt.com

[20] XML Diff and Merge Tool by IBM. http://alphaworks.ibm.com/tech/xmldiffmerge

[21] VM Tools by VM Systems. http://www.vmguys.com/vmtools/.

[22] XML TreeDiff by IBM. http://alphaworks.ibm.com/tech/xmltreediff

[23] DeltaXML by Mosell EDM ltd. http://www.deltaxml.com

[24] Andreas Laux and Lars Martin. XUpdate Working Draft. XML:DB Initiative, September 2000.

[25] XML:DB Initiative for XML Databases. http://www.xmldb.org

[26] Reference implementation for XUpdate. http://www.xmldb.org/xupdate/index.html

[27] Michael J. Pont. Software Engineering with C++ and CASE Tools. 1996. ISBN 0-201-87718-X.

[28] Myers E.W. An $o(nd)$ difference algorithm and its variations. 1986. Algorithmica 1, pages 251-266.

[29] XML Path Language (XPath). World Wide Web Consortium, November 1999. http://www.w3.org/TR/xpath

[30] Namespaces in XML. World Wide Web Consortium, January 1999. http://www.w3.org/TR/REC-xml-names/

[31] Extensible Stylesheet Language Transformation (XSLT). World Wide Web Consortium, November 1999. http://www.w3.org/TR/xslt

[32] Open Source Initiative. http://www.opensource.org

[33] SourceForge. Open source development environment. http://sourceforge.net

[34] The GNU General Public License. Free Software Foundation, June 1991. http://www.gnu.org/licenses/gpl.txt