Helsinki University of Technology
Department of Computer Science
Laboratory of Information Processing Science

Tancred Lindholm

A 3-way Merging Algorithm for Synchronizing Ordered Trees — the 3DM merging and differencing tool for XML

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science.

Helsinki, September 13, 2001

Supervisor	Professor Martti Mäntylä
Instructor	M.Sc. Mervi Ranta

HELSINKI UNIVERSITY OF TECHNOLOGY | ABSTRACT OF MASTER'S THESIS

Author:	Tancred Lindholm	
Title:	A 3-way merging algorithm for synchronizing ordered trees — The "3DM" merging a differencing tool for XML	
Date:	September 10, 2001	**Number of pages:** 128
Department:	Computer Science	
Proffessorship:	Tik-76 Software Engineering	
Supervisor:	Professor Martti Mäntylä	
Instructor:	M.Sc. Mervi Ranta, Helsinki University of Technology	

Keeping data synchronized across a variety of devices and environments is becoming more and more important as the number of computing devices per user increase. Of particular interest are situations when the sets of data that need to be synchronized have structure, but not the exact same stucture.

In the thesis we approach these situations trough a number of use cases, which are set in a future, more ubiquitous, computing environment. The use cases are subsequently analyzed, and in combination with the characteristics of a ubiquitous computing environment used to derive requirements for a synchronization tool. Although the focus is on future computing environments, we find that a synchronization tool fulfilling the requirements is well suited for present-day synchronization tasks as well.

We find that the requirements call for a tool capable of performing a 3-way merge of general ordered trees without any additional tree metadata, such as edit histories or unique node identifiers, that describe how the trees participating in the merge are related. The main research problem of the thesis is to design such an algorithm, given that no suitable algorithms exist.

The design of the algorithm is preceded by stating a definition of desired merging behaviour derived from the use cases as well as a relatively large number of small hand-written merging examples. Further on, the design task is divided into the design of a tree matching algorithm, and an algorithm for merging matched trees. The matching algorithm also gives us the ability to easily find the difference between ordered trees.

The designed merging, matching and differencing algorithms for ordered trees are implemented as a merging and differencing tool for XML, and their functionality is verified against the use cases as well as the merging examples. The evaluation of the algorithms shows promising results regarding the applicability of the tool to real-world situations.

Keywords: synchronization, ubiquitous computing, 3-way merging, ordered tree, differencing, XML

Tekijä: Tancred Lindholm

Työn nimi: Kolmen synkronoitavan puun yhdistämisalgoritmi ja sen toteutus XML:n yhdistämis- ja vertailutyökaluna ”3DM”

Title: A 3-way merging algorithm for synchronizing ordered trees — The ”3DM” merging a differencing tool for XML

Päivämäärä: 10.9.2001 **Sivumäärä:** 128

Osasto: Tietotekniikan osasto

Professuuri: Tik-76 Ohjelmistotuotanto

Työn valvoja: Professori Martti Mäntylä

Työn ohjaaja: DI Mervi Ranta, Teknillinen Korkeakoulu

Tiedon synkronointi muuttuu yhä tärkeämmäksi henkilökohtaisten laitteiden määrän ja teknisten ympäristöjen moninaisuuden lisääntyessä. Erityisen kiinnostavia ovat synkronointitilanteet, joissa synkronoitavat datajoukot ovat rakenteellisia, mutta niiden rakenne ei ole identtinen.

Rakenteellisten tietojoukkojen synkronointiongelma lähestytään työssä lähitulevaisuuteen sijoittuvien käytöesimerkkien (use case) avulla. Esimerkit ja ubiquitous computing -ympäristö, johon esimerkit sijoittuvat, analysoidaan ja analyysin tuloksena saadaan vaatimuksia synkronointityökalulle. Huomataan että näiden vaatimuksien mukaista työkalua voidaan myös hyvin soveltaa ajankohtaisiin rakenteellisiin synkronointiongelmiin.

Eräs vaatimusten mukainen synkronointimenetelmä on kolmen puun yhdistäminen (3-way merge), jossa ei oleteta mitään puihin liittyvän metadatan, kuten editointihistorian tai puiden noodien uniikin nimeämisen, olemassaoloa. Työn päällimmäinen tutkimusongelma on suunnitella puiden yhdistämisalgoritmi, joka toimii näillä oletuksilla.

Algoritmin suunnittelua varten määritellään haluttu yhdistämiskäytös käyttöesimerkkien sekä suurehkon joukon käsin tehtyjen pienten puiden yhdistämisesimerkkien avulla. Suunnittelutehtävä jaetaan puiden yhteensovittamiseen (tree matching) ja yhteensovittujen puiden yhdistämiseen (merging of matched trees). Puiden yhteensovittamisen avulla voidaan myös helposti tuottaa kahden puun ero (diff).

Työssä suunnitellut algoritmit on toteutettu XML-tiedostojen yhteesovittamis- ja vertailutyökaluna “3DM“. Algoritmien toimivuus testattiin suunnittelussa käytettyjen käyttöesimerkkien ja yhdistämisesimerkkien avulla ja se todettiin hyväksi. Testituloksien mukaan työkalu soveltunee suoraan myös reaalimaailman rakenteellisiin synkronointiongelmiin.

Avainsanat: synkronointi, ubiquitous computing, kolmen puun yhdistäminen, puurakenteet, tiedostojen vertaileminen, XML

Acknowledgements

This work would not have been possible without the support of the researchers at Helsinki University of Technology. I especially wish to thank my supervisor, professor Martti Mäntylä and my instructor, M.Sc. Mervi Ranta, for giving me the freedom to carry out my ideas. I am also grateful to the researchers in the Product Modeling and Realization Group and the UbiServices/Between project who have provided me with inspiration and valuable feedback.

I wish to thank my parents, Christel and Mikael, and my brothers for giving me such a harmonious environment to grow up in, and for their never-ending support for my interest in computers.

Special thanks go to the open-source community for providing me with such wonderful software as LyX, LaTeX and Linux.

Finally, I would like to thank my friends (you know who you are) for making the story of my life so much more interesting, and for putting up with me during the finalization of this thesis.

September 13, 2001 in Helsinki, Finland

Tancred Lindholm

Contents

1 Introduction **1**
1.1 Preliminaries on version control, trees and XML 2
1.1.1 Fundamental concepts in version control 2
1.1.2 Trees 4
1.1.3 XML - the extensible markup language 6

2 Synchronization Use Cases **8**
2.1 Use case 1: A shared, intelligent shopping list 9
2.2 Use case 2: Related documents stay up-to-date 11
2.3 Use case 3: Updating an annotated illustration in a document 11
2.4 Use case 4: Merging changes from several authors 14
2.5 Use case 5: Maintaining several versions of a web page 17

3 The Structural 3-way Merge Problem **21**
3.1 Data synchronization 21
3.2 Ubiquitous computing in the UbiServices project 23
3.3 Thesis problem statement 26
3.4 Merge tool requirements 29

4 Work on Synchronization, Merging and Differencing **31**
4.1 Overview of synchronization approaches 31
4.2 Differencing tools 33
4.3 Merge tools 35
4.4 Algorithmic research 36
4.4.1 Tree differencing algorithms 36
4.4.2 Edit distance and scripts 37
4.4.3 Tree Matchings 37
4.4.4 Tree merging algorithms 39
4.5 Discussion of related work 41

5 Defining Tree Matching and Merging **44**
5.1 Desired behavior of the 3-way merging tool 45
5.1.1 Merge cases 45
5.1.2 Functional requirements for 3-way merging 45
5.2 Tree matching definitions and requirements 47
5.3 Tree merging definitions and requirements 52

5.4 The architecture of 3DM . . . 56
5.4.1 Tracing the merge and producing readable diffs . . . 57

6 The Tree Matching Algorithm **59**
6.1 A heuristic tree matching algorithm . . . 59
6.1.1 Measuring node similarity . . . 60
6.1.2 Subtree matching: `procedure matchSubtrees` . . . 63
6.1.3 Removing small copies: `procedure removeSmallCopies` . . . 65
6.1.4 Matching similar unmatched nodes: `procedure matchSimilarUnmatched` . 66
6.1.5 Setting the types of the match: `procedure setMatchTypes` . . . 68
6.2 Analysis . . . 68
6.2.1 The set and map data structures . . . 68
6.2.2 Complexity analysis of the matching algorithm . . . 69

7 The Tree Merging Algorithm **75**
7.1 The merging algorithm . . . 75
7.1.1 Generating the merge pair list . . . 77
7.1.2 Additional conflict checking . . . 84
7.1.3 Merging the contents of nodes u_i and v_j . . . 85
7.1.4 Updating the cursors C_1 and C_2 . . . 86
7.2 Analysis of the algorithm . . . 87
7.2.1 Correctness of the algorithm . . . 88
7.2.2 Complexity analysis . . . 91
7.2.3 Algorithm termination . . . 96

8 The Tree Differencing Algorithm **98**
8.1 Algorithm design considerations . . . 98
8.2 The algorithm . . . 99
8.2.1 Encoding a matching as a diff . . . 99
8.2.2 Generating the matching between T_1 and T_B. . . . 101
8.2.3 Patching: Regenerating T_1 from T_B and the diff . . . 103
8.3 Algorithm complexity . . . 103

9 The Implementation of 3DM **105**
9.1 Expressing and resolving conflicts . . . 105
9.2 Logging merge operations . . . 106
9.3 Invoking 3DM . . . 110
9.4 Overview of the Java implementation . . . 110
9.5 Unimplemented XML features . . . 111

10 3DM Evaluation **113**
10.1 Merge cases . . . 113
10.2 Use cases . . . 114
10.2.1 Use case 1: The shopping list . . . 114
10.2.2 Use case 2: Related documents stay up-to-date . . . 116
10.2.3 Use case 3: Updating an annotated illustration in a document . . . 117
10.2.4 Use case 4: Merging changes from several authors . . . 117

10.2.5 Use case 5: Maintaining several versions of a web page 118
10.3 Conclusions of the evaluation 120

11 Conclusions and Future Work 121

A Family Virtanen Home Scenario 129

B Use Case 1: Shopping List 133

C Use Case 2: Related Documents Stay Up-to-date 136
Profile, base and updated version 136
Homepage, after updated profile has been applied 137
Homepage, after favorite music has been updated 138
Base version of the Book of friends, rendered view in section 2.2 139
Updated version of the Book of friends 139
Original homepage of aunt Maija, rendered view in section 2.2 140

D Use Case 3: Updating an Annotated Illustration 141
Draft modem drawing, SVG source (rendering in section 2.3) 141
Completed modem drawing, rendering and SVG source 142
Base version of manual, XHTML source (rendering in section 2.3) 143
Manual with updated drawing, XHTML source (rendering in section 2.3) 144

E Use Case 4: Document Review 145

F Use Case 5: Maintaining Several Versions of Web Pages 157
XHTML source for base complex version, rendering in section 2.5 157
XHTML source for simple base version 159
Rendering of updated simple version 160
Rendering of manually merged complex version 160
XHTML source for updated simple version 161
XHTML source for merged complex version 161

G Merge Cases 163

H 3DM Source Code 172
BaseNode.java 173
BranchNode.java 173
ConflictLog.java 174
Diff.java 175
DiffMatching.java 176
EditLog.java 176
MatchArea.java 177
MatchedNodes.java 177
Matching.java 178
Measure.java 181
Merge.java 182
Node.java 187

NodeFactory.java 188
ParseException.java 188
Patch.java 188
PathTracker.java 189
TreeDiffMerge.java 190
TriMatching.java 191
XMLElementNode.java 192
XMLNode.java 192
XMLParser.java 193
XMLPrinter.java 193
XMLTextNode.java 194

Index of Definitions and Terms **195**

List of Notations

$R, a \ldots z, a_1, \ldots$	tree nodes
(n, m)	edge
$T, T_1, \ldots$	trees
$\mathbf{n}$	child list of node $\mathbf{n}$
$\langle T_1, T_2 \ldots \rangle$	forest
T_n	subtree rooted at n
$\{a, \ldots\}$	set
$S_1 \times S_2$	Cartesian product of sets
$T \ominus T'$	tree difference
$\mathcal{E}, \mathcal{E}_1 \ldots$	edit script
$\mathcal{M}, \mathcal{M}_1 \ldots$	tree matching
$\mathrm{function}(\cdot)$	function
$predicate(\cdot)$	predicate
$C_1, \ldots$	tree cursor
$M_1, M_2, \ldots M_1^D, \ldots$	merge list
$\{n, m\}$	merge pair
$\mathbf{M}$	merge pair list

See the index for definitions.

List of Abbreviations

3DM	3-way merging and differencing tool
API	application programming interface
BFS	breadth-first search
CSCW	computer supported cooperative work
DFS	depth-first search
DTD	document type definition
GSM	global system for mobile communication
GUI	graphical user interface
HTML	hypertext markup language
HTTP	hypertext transfer protocol
IETF	internet engineering task force
LAN	local area network
PDA	personal digital assistant
SGML	standard generalized markup language
SVG	scalable vector graphics
UMTS	universal mobile telecommunications service
URL	uniform resource locator
WWW	world wide web
W3C	world wide web consortium
XHTML	reformulation of HTML in XML
XML	extensible markup language

Chapter 1

Introduction

Keeping data up-to-date across a variety of devices and environments is becoming more and more important. The typical user of information technology no longer has access to single computer, but several, in the form of home and office workstations, personal digital assistants (PDAs) and mobile phones. Using several devices and environments has lead to the spreading out of the user's digital environment: his calendar may reside in the PDA as well as on the home and office PCs, his phone book in his mobile phone and office PC and his unfinished novel as an HTML document on the Internet as well as a word processor file on his home PC.

Having information in several places leads to the question of how to keep it all synchronized. Appointments added to the calendar on the PC should also be visible on the PDA. Changes made to the novel in the word processor should also appear in the HTML version, for his friends to review.

In the thesis we approach the problem of data synchronization by designing and implementing a tool for automatic integration of different versions of a tree data structure. As an example of such integration, consider a document edited by two reviewers: The first reviewer moves a paragraph, while the second reviewer corrects a spelling error in the same paragraph. When automatically integrating these two versions, we would like to get a version where the paragraph has been moved and the spelling error is corrected.

We will present an algorithm for automatic integration of trees, i.e. tree merging, as well as its implementation in the form of a tool for merging and differencing of XML (extensible markup language) documents, called "3DM" (3DM stands for 3-way merging, Differencing and Matching). We will not address synchronization without integration, i.e. the task of exactly replicating a set of data from one peer to another, nor will we consider issues regarding synchronization policies or how data is transported between peers that need to synchronize.

The thesis is a contribution to the UbiServices/Between research project [Btw] on future ubiquitous and mobile computing services, whose objective is to generate service concepts for the future nomadic users. In the project, we are not interested just in bringing current Internet services to the mobile networks but rather trying to predict and discover potential future mobile and ubiquitous services. The data synchronization problem is considered in this context, and thus we will focus on a future ubiquitous and mobile computing environment, an environment in which we feel that the problem will become increasingly more important.

The starting point for the algorithmic design and the implementation of the tool is a set of use cases, presented in chapter 2. The use cases were devised in the context of a fictious scenario that the research team of the UbiServices project invented in order to have a basis for discussing future ubiquitous computing services and how to classify the potential directions for research. The use cases are elaborated to the point where they can be used to test and verify the implementation of the 3DM tree merging tool.

In chapter 3, we state the definition of data synchronization used in the thesis and present our vision of the ubiquitous and mobile computing environment, which in combination with the use cases allows us to derive and formulate the research problems of the thesis, the most important of which being the design and implementation of 3-way merging tool for ordered trees, which fulfills a set of requirements derived from the use cases and the environment of the tool. Although the requirements are derived from an assumed future computing environment, they are very much valid for present-day environments, an thus the tool is immediately useful as a general tool for XML tree merging and differencing. Existing synchronization technologies and algorithmic research related to the construction of the tree merging algorithm is discussed in chapter 4.

We approach the tree merge problem in chapter 5, by dividing it into two subproblems: tree matching and merging. We then continue by first informally, and then more rigidly defining the desired behavior of the merge tool. In addition to the use cases, we use a quite large number (42) of smaller test cases as a basis for the definition of the desired merging behavior. In the process several useful definitions related to tree matching and merging are made.

The overall architecture of the tool is presented in section 5.4, where it is divided into matching, merging and differencing modules. The matching algorithm is presented in chapter 6, the merging algorithm in chapter 7 and the differencing algorithm in chapter 8. The complexities of the algorithms are analyzed, and in addition some relevant properties of the merging algorithm are proved. The implementation of the tool is presented in chapter 9.

In chapter 10 we evaluate the tool on the use cases as well as on the test cases. Conclusions and future work is presented in chapter11.

1.1 Preliminaries on version control, trees and XML

Before we begin reviewing related work we will introduce some important terminology and definitions that will be needed throughout the thesis.

1.1.1 Fundamental concepts in version control

Version control deals with the managing of different versions and variants of data files. The field is large, and hence we will only present some fundamental concepts, that will be needed in the thesis. The terminology presented in this section originates from the well-known Concurrent Version System [Berl90, Ced93]. To illustrate the concepts we will use a hypothetical text file `panic.c`that contains the source code for a simple “Don’t Panic” program.

During the writing of `panic.c` several increments of the file may be saved: the initial program skeleton which does nothing, the first version which merely outputs “Don’t Panic” on the screen, and the final version that shows the greeting in large, friendly letters. We call those increments of the program that have been stored (e.g. by creating files `panic-1.c`, `panic-2.c` etc.) *revisions* of the program. Different revisions of a file are typically identified by a *revison number*. CVS for instance uses 1.1, 1.2, ... We will refer to the three revisions of `panic.c` described above as revisions 1, 2 and 3.

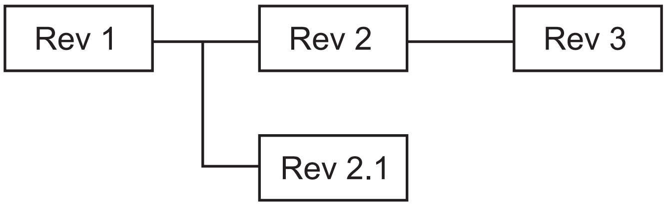


Figure 1.1: Example of revision tree. The example shows the revision tree of `panic.c` after a parallel revision to revision 2 has been created.

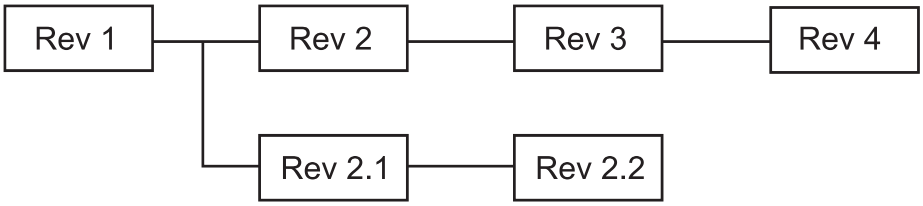


Figure 1.2: Revision tree of `panic.c` after revisions 2.2 and 4 have been created.

Suppose we in addition to having the greeting displayed in large bold letters want to create a variant of the program that shows the greeting in huge letters of fire, representing a parallel development line of of `panic.c` revision 2. We call such a parallel development of a revision a *branch*. A single revision may be the basis of several branches, and the branches may contain revisions local to the branch, as well as sub-branches. This gives the notion of a *revision tree*, as illustrated in figure 1.1. We identify the second variant of `panic.c` with the revision number 2.1 (meaning the first revision based on revision 2). The *branch point* of two revisions on different branches is the first common revision of the revisions, when traversing the revision tree towards the root. As an example, the branch point of revisions 2.1 and 3 is revision 2. Furthermore, we say that files that share a common revision at some point in the revision tree are branches of that revision. For instance, revisions 3 and 2.2 of `panic.c` are branches of revision 2. Here, the term *version* is used as a general term for revisions, that may be branches or not, and usually has some semantic meaning to the user. For instance, in software development, typically a lot of revisions of the source files are generated, but we only speak of versions 1.0 and 2.0 of the software.

In continued development of the program we add a `--politeness` option to revision 2.1, that apologizes the user for the inconvenience after having displayed the fiery message, thus creating revision 2.2. This clearly is a very useful feature, which we would like to include in the first variant of the program as well. That is, we would like to *merge* the changes made between revisions 2.1 and 2.2 to revision 3. The result, a version that is polite and uses large, friendly letters becomes revision 4. The revision tree now looks as illustrated in figure 1.2.

In order to perform the merge we need to 1) extract the changes between between revision 2.2 and 2.3 and 2) apply the changes to revision 3 of `panic.c`. The process of extracting the changes between two revisions (or any arbitrary files for that matter) is here referred to as *differencing* and the output of the differencing process as a *diff*. Figure 1.3 shows two files, and the diff produced by the standard UNIX differencing utility `diff`. The diff can be understood as a sequence of edit operations that transforms the first input file to the second.

Applying the changes (step 2) is accomplished by running a program that executes the edit operations implied by the diff on the target file, in this case revision 3 of `panic.c`. This process is referred to as *patching*. The UNIX utility for this purpose is, quite aptly, called `patch`.

```
LyX is an
excellent
word processor.
```

File 1

```
LyX is an
excellent open-source
word processor.
Visit www.lyx.org!
```

File2

```
2c2
< excellent
---
> excellent open-source
3a4
> Visit www.lyx.org!
```

Diff

Figure 1.3: Two files and their difference, as generated by the `diff` utility. The `2c2` line in the diff means that line 2 should be replaced (the old and new line follow), and the `3a4` line means that the following line in the diff should be appended after line 3.

1.1.2 Trees

Assume a set of *nodes* V and a set $E \subset V \times V$ of edges connecting the nodes. If $(u, v) \in E$ we say that (u, v) is an *edge* and that u is the *parent* of v. $T = (V, E)$ is a *rooted tree* if the set of edges satisfy the following conditions

1. Exactly one node $R \in V$ has no parent. This node is the *root* of the tree.

2. Every node, except the root, has exactly one parent.

3. Each node in V is reachable via edges from the root.

The *children* of a node u are all nodes whose parent is u. The *descendants* of u are the children of u, as well as the descendants of the children. If v is a descendant of u, then u is the *ancestor* of v. A node that has no children is called a *leaf node*. A nodes that is not a leaf node (and thus has children) is called an *interior node*.

A *path* in the tree from v_1 to v_n is the sequence of edges (v_1, v_2), $(v_2, v_3), \ldots, (v_{n-1}, v_n)$, where each pair $(v_m, v_{m+1}) \in E$. The *length* of a path is the number of edges in the path. We can now define the *height* of a tree T to be longest path in T increased by one. Each node in the tree also has a *depth*, which is given by the length of the path to the root increased by one. The root thus has a depth of 0. The nodes at depth d form *depth level* d of the tree.

We say that a tree is *ordered* if the children $v_1 \ldots v_k$ of a node u are uniquely numbered from 1 to k. The *child list* of the node u, denoted by $\mathbf{u}$, is then the list of the children of u in the order $v_1, v_2, \ldots v_k$. The notation $\mathbf{u}(i)$ denotes the item v_i from $\mathbf{u}$. For ordered trees, we define the *left sibling* of the child v_l, $l > 1$ to be v_{l-1} and the *right sibling* of v_l, $l < k$ to be v_{l+1}. In figures, the children are drawn left-to-right according to their numbering, with the node having number 1 as the leftmost node. A tree that is not ordered is *unordered*.

The *subtree* T_u of T is the tree rooted at $u \in V$. A collection of the trees $T_{i_1}, \ldots, T_{i_k}$ is called a *forest*, denoted by $\langle T_{i_1}, \ldots, T_{i_k} \rangle$. A tree is *labeled* if there is a labeling function $f_l : V \to L$, where L is a set of labels using some alphabet.

Throughout the text there are several examples of trees. Trees are written using the following notation:

1. Nodes are labeled $R, a, b, c, \ldots, a_1, \ldots, a', \ldots$ The root node is usually labeled with R.

2. A tree consisting of one node is denoted by the node label.

3. A tree T with more than one node is denoted by $(a; T_{a_1}\, T_{a_2} \ldots T_{a_n})$, where T_{a_n} are the subtrees rooted at the children a_n of a.

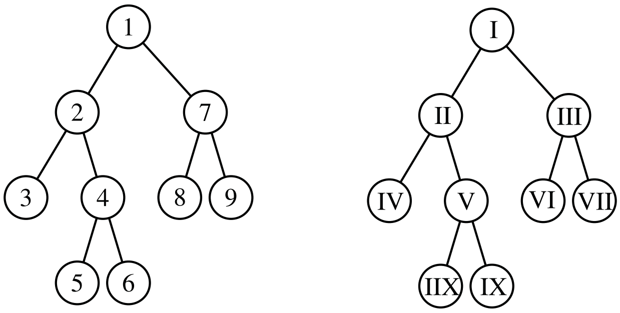


Figure 1.4: DFS and BFS traversals of a tree. The Arabic numbers shows the DFS traversal order and the Roman numbers the BFS traversal order.

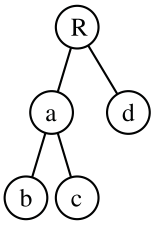


Figure 1.5: Example tree.

Suppose $T_1 = (R;\, a\, b)$ and $T_2 = (R;\, a\, c)$. To distinguish the node a in T_1 from the equally labeled node in T_2 we use the notations $T_1(a)$ and $T_2(a)$.

There are several ways of traversing the nodes in a tree. In the thesis we will use *depth-first search* (*DFS*), where a node is visited after all the nodes in the subtree rooted at the node's left sibling have been visited, and *breadth-first search* (*BFS*), where the nodes at depth level n are visited in left-to-right order before the nodes at depth level $n + 1$. Figure 1.4 illustrates depth- and breadth-first search of a tree.

As an example, consider the tree depicted in figure 1.5. The nodes of the tree are labeled with $R, a, \ldots, d$, with R being the root node. Using the notation for trees described above, the tree is written as $(R;\, (a;\, b\, c)\, d)$. The children of R are a and d, and the parent of b and c is a. The ancestors of b are a and R, and the descendants of a are b and c. The subtree T_a is $(a;\, b\, c)$. The nodes R and a are interior nodes and b, c and d are leaf nodes. The height of the tree is 3 and the depth of a and d is 2. The right sibling of a is d, and the left sibling of d is a.

The difference between two trees T_1 and T_2 will be denoted by $T_1 \ominus T_2$. The notation does not specify how the difference is expressed.

The algorithms presented in this thesis are designed to process hypertext parse trees, such as those produced when parsing XML, HTML and SGML. We accommodate for this in our tree model by defining that each node v has an associated *node content*, denoted by $\mathrm{content}(v)$. The node content consists of a type field, a name field, a text field and a list of attributes and their associated values, denoted by $\mathrm{type}(v)$, $\mathrm{name}(v)$, $\mathrm{text}(v)$, $\mathrm{atr}(v, i)$ and $\mathrm{val}(v, attributeName) = \mathrm{val}(v, \mathrm{atr}(v, i))$. When labeling nodes, the labels are the same iff the contents of the nodes agree, i.e.

$$f_l(v_1) = f_l(v_2) \Leftrightarrow \mathrm{content}(v_1) = \mathrm{content}(v_2) \tag{1.1}$$

Unless stated otherwise trees are labeled and ordered, their nodes have an associated content and the labeling function satisfies proposition 1.1.

```
<?xml version="1.0"?>
<paragraph author="ctl" date="11/7/2001">
  Hello,
  <bold>
    world!
  </bold>
  <anchor name="end-of-paragraph" />
</paragraph>
```

Figure 1.6: A sample XML document

1.1.3 XML - the extensible markup language

The extensible markup language (XML) [W3C01e] provides a much needed non-proprietary universal format for sharing hierarchical data among different software systems, and does so in a user-friendly manner. XML is a verbose plaintext format, making it robust, platform-independent and legible[1] to humans without additional tools. XML was designed to be a easily deployable subset of the Standardized Generalized Markup Language [SGML], which is widely used by government agencies and major corporations to ensure interoperability and persistent data accessibility.

XML is currently being heavily pushed by the industry and community as the *lingua franca* for data exchange on the Internet. XML is supported by the World Wide Web Consortium [W3C] as well as major corporations such as Microsoft, Sun Microsystems and IBM. It can also be noted that XML is one of the principal components of Microsoft's much-touted .net initiative.

XML is a *meta language*, meaning that it is used to describe document languages, which in turn describe the structure of documents. An example of an XML-based document languages is XHTML. There exists a myriad of XML-based languages, of which a large fraction are created ad-hoc when the need to encode structural data emerges. The shopping list, which will be presented in use case 1, is an example of such an ad-hoc XML language.

The main parts of an XML document is the prolog and the root element. An XML *element* (including the root element) consists of either an empty-element *tag* or matching start and end tags, in between which there may be character data and other elements (as well as some other constructs that are of lesser importance in the context of this thesis). Each element may have list of associated attributes and values, which are listed in the start or empty-element tag. Tags and attributes are usually given meaningful names, such as `<paragraph>` or `<chapter>`, making XML documents self-explanatory. [W3C01e]

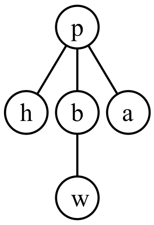
For instance, consider the XML document in figure 1.6 . Here the root element is a paragraph, indicated by the `<paragraph>` and `</paragraph>` start and end tags. The `</paragraph>` tag has two attributes associated to it: `author` whose value is `ctl` and `date` whose value is `11/7/2001`. The content of the root element is the word `Hello,` as well as the `bold` and `anchor` elements. The `anchor` element is an example of an empty-element tag (note the `/` at the end of the tag).

We will map XML documents to trees according to the following rules:

1. Elements and character data are mapped to nodes.

2. The children of a node n are the elements and character data contained in the element corresponding to n.

[1] One may argue about the legibility of XML, but it certainly is more legible than raw binary data.



Label	Type	Name	Text	Attributes and values
p	element	paragraph		author=ctl date=11/7/2001
h	text		Hello,	
b	element	bold		
w	text		world!	
a	element	anchor		name=end-of-paragraph

Figure 1.7: The tree corresponding to the XML document in figure 1.6.

3. The content of a node corresponding to an element has a type value of `element`, the name field is initialized to the name of the element (e.g. "`paragraph`" form the example above), and the list of attributes and values are initialized with the attributes and values of the element. The text field is left empty.

4. The content of a node corresponding to character data has a type value of `text`, and the text field is initialized with the character data. The other fields are left empty.

5. Content that is not character data nor elements (such as comments) is not mapped to the tree.

Nodes, whose type field has the value `element` are called *element nodes*, and those whose type field is `text` are called *text nodes*. Figure 1.7 shows the tree corresponding to the XML document in figure 1.6.

In the use cases we will make frequent use of *XHTML*. XHTML is the World Wide Web Consortium's [W3C] recommendation for the latest version of HTML, and is essentially a reformulation of HTML version 4 in XML [W3C01b]. As XHTML is an XML application it is parseable by any conforming XML parser, yet it is similar enough to HTML 4.0 to render without problems in current WWW browsers, such as Netscape Navigator, Internet Explorer or Mozilla. These properties make XHTML well suited for presenting several text-based use cases in this thesis.

A way of addressing nodes in an XML document tree from another XML document will come in handy on several occasions. We will use a very simple subset[2] of the *XPath* notation, defined by the W3C in [Cla99]. Nodes are identified using absolute paths, consisting of an initial / (slash), and any number of steps, each step separated by slashes. Each step indicates the position of the child node to traverse on the path, the first child having position 0. For instance, in figure 1.6 the root node (`<paragraph>`) is identified by the path `"/"`, the `<anchor>` node by `"/2"` and the the `world!` text node by `"/1/0"`.

[2]The path expressions used are valid XPath expressions

Chapter 2

Synchronization Use Cases

We approach the topic of the thesis by by looking at five everyday situations, or use cases, where data synchronization is needed. These use cases will then be used throughout the thesis to aid in the design of the 3DM tool; in the beginning by providing requirements for the tool, and in the end by providing examples of ideal synchronization, against which the performance of 3DM can be evaluated. The use cases also justify the topic of the thesis, by showing the need for a synchronization tool like 3DM. The use cases are based on the following three sources:

- The family Virtanen home scenario.
- Preliminary results of the validation of ubiquitous service ideas performed by Tomi Kankainen and Tapio Lehtonen in the context of the UbiServices project.
- Identification of common tasks in the work environment of the UbiServices research group that may benefit from structural synchronization.

The use cases have been elaborated to the point where the synchronization tasks are clearly identified. To further illustrate the case, and to allow the 3DM tool to be validated against the use cases, files containing realistic data has been created for each case. The files include all the necessary input for the synchronization tool, as well as the desired result of the synchronization.

All use cases are set in the context of the family Virtanen scenario, the full text of which is available in appendix A. The Virtanen family home scenario has the following participants:

Jussi Virtanen technical writer and eager chess player. Jussi is married to

Anna Virtanen, a journalist at a mid-size newspaper. They have two children:

Liisa Virtanen their 15-year-old daughter and

Kalle Virtanen their 7-year-old son.

Maija Mellunmäki also known as “aunt Maija”. A good friend of the Virtanens’.

The scenario describes a typical December day in the life of the Virtanens, sometimes in a not too distant future when computing has become more ubiquitous, including smoothly working data synchronization.

In the morning everybody is awakened according to their personal preferences. They rise, get dressed and eat breakfast, after which they leave for school and work. During the day Mrs. Virtanen invites aunt Maija and her husband to come and visit them, making it necessary to get the house cleaned up and change the dinner plans. Mr. Virtanen offers help with the cleaning and Liisa helps with the groceries.

In the late afternoon the children and Mr. Virtanen return home from work and begin preparing for the guests. Mrs. Virtanen is supposed to give aunt Maija and her husband a ride to the Virtanen residence, but unfortunately her car breaks down. This complicates the transportation a bit, and causes Mrs. Virtanen and the guests to arrive late.

In spite of the broken car, the visit turns out quite nicely. The food is good, and the home entertainment system is further able to increase the comfort. The visit ends at around 10 pm, when aunt Maija and her husband return home.

2.1 Use case 1: A shared, intelligent shopping list

The Virtanen family have a common shopping list, to which any family member may add things to buy. There is also a shopping service that maintains an automatically generated "suggested shopping list". This list basically contains the same items as the shopping list, but also includes product prices, where to buy the product, and additional suggestions made by the shopping service: new items as well as comments and changes in quantities. See the L0 and L1 in appendix B for an example of a common shopping list (L0) and a suggested list (L1) based on L0.

During the day described in the home scenario several interactions with the shopping lists are made:

1. In the morning, after breakfast there is no orange juice left in the refrigerator. The shopping service adds 2 liters of orange juice to the suggested list. The service also notices that there is no toilet paper left, and adds that to the list as well.

2. When traveling to work on the bus, Mr. Virtanen browses his incoming messages and notices one about his son's school Christmas party. The message is from the teacher, and suggests that everybody bring a cheap Christmas present with them. Mr. Virtanen thinks this is a good idea, and adds a Calvin & Hobbes comic album to the shopping list.

3. During lunch break at school Liisa comes to think of, out of the blue, that she has not had apples for ages. She adds some to the shopping list.

4. Mrs. Virtanen reviews the shopping list from work in order to get it ready before her daughter leaves school, as she wants her to do the shopping. She prefers to use the suggested list, as the shopping service usually makes quite reasonable suggestions. The suggestions made by the service are clearly marked out among the items added by the family members, so that Mrs. Virtanen can review them easily.

   This time the shopping service was especially useful, as it pointed out that aunt Maija, who is visiting the Virtanens today, is a vegetarian, and would probably object to the smoked salmon Mrs. Virtanen intended for dinner. Mrs. Virtanen removes the salmon, adds some lettuce and a princess cake, and makes some adjustments to the quantities of vegetables

5. On her way home from school Liisa goes shopping.

Event	Description	Synchronization task	Inputs	Out	C	S	J	L
	Initial state	-	-	L0	L0	L0	L0	L0
1	Service adds items	- (None, since nobody has updated the list)	L0	L1	L0	L1	L0	L0
2	Mr. V adds present	-	L0	L2	L0	L1	L2	L0
3	Liisa adds apples	-	L0	L3	L0	L1	L2	L3
4	Ms. V reviews list	All edits to the home list are combined.	L0,L2,L3	L4	L4	L1	L2	L3
5		The suggested list is updated, as a result of the home list changing.	L0,L4,L1	L5	L4	L5	L2	L3
6		The shopping service makes some suggestionsand fills in empty fields (e.g. prices)	L5	L6	L4	L6	L2	L3
7	Ms. V is done reviewing	Ms. V changes the suggested list accordingto her preferences	L6	L7	L7	L7	L2	L3
8	Liisa goes shopping	Liisa gets the latest shopping list.	-	-	L7	L7	L2	L7

Table 2.1: Shopping list synchronization events. The entries in the inputs and output columns refer to the names of the XML files listed in appendix B. The four last columns are: C = common shopping list, S = suggested shopping list, J = shopping list in Mr. Jussi Virtanen's PDA, L = shopping list in Liisa's PDA.

As the synchronization tasks in the case are not explicitly mentioned some elaboration on the use case is needed to extract them. Let us assume a simple synchronization policy for maintaining the shopping list:

- changes are instantly propagated between devices connected by a fast, cheap network[1]
- a network connection is normally not established to bring the list up to date when the network connection is slow and expensive (e.g. a wireless connection), unless the latest version is explicitly needed.

In this particular case, Mr. Virtanen and his daughter operate in slow network environment, whereas Mrs. Virtanen, the common shopping list, and the shopping service are connected to a fast network. Thus, the common and the suggested shopping list (produced by the shopping service) are always kept in sync, and Mrs. Virtanen can edit the common and suggested lists directly. The lists of Mr. Virtanen and Liisa are synchronized only when necessary. This policy leads to the synchronization events listed in table 2.1. For instance, the fourth event in the table describes how the edits Mr. Virtanen and Liisa have made to their local copies of the shopping list are integrated to the common shopping list. The locally modified version on Mr. Virtanen's PDA is L2 and the one on Liisa's PDA is L3. Both L2 and L3 are branches of the initial shopping list, L0, and thus we have a 3-way merging scenario of L0, L2 and L3. The result of the integration is named L4.

The various stages of the common and suggested shopping lists, written in self-explanatory XML (using tags such as `<item>`, `<price>` and `<quantity>`) are in appendix B.

[1]The network types described here correspond to the notions of tight and weak coupling, which we will define in section 3.1

2.2 Use case 2: Related documents stay up-to-date

Let us have a closer look at how information about aunt Maija is accessed by different persons and from different sources in the home scenario. We will see that the same pieces of information are copied across many documents, and there is a need to automatically propagate changes in these pieces of information to the copies.

Aunt Maija discovered the World Wide Web a few months ago, and promptly decided to make a home page of her own (see figure 2.1). Her home page includes, among other things, her telephone number and address as well as a list of her favorite music.

The Virtanens have a “Book of friends”, where they keep contact information, birthdays and notes about their friends. On the page about aunt Maija (see figure 2.2) there’s a note about the esthetics of a vase she gave a few years ago and, copied from her home page, the list of her favorite music as well as her contact information.

Aunt Maija moved to a new house a few weeks ago, changing her address as well as her telephone number. Informing all your aquintances about the new address used to be a real problem, but nowadays it usually works quite smoothly. For instance, as soon as aunt Maija updated the address and telephone number in her profile, the changes propagated to her home page. This update of her home page was in turn propagated to every document around the world that includes a part of it (typically her contact information), including the Virtanens’ Book of friends.

Maija’s taste of music is constantly changing, and from time to time she even remembers to update the list of her favorite music on her home page. Last time she decided that Toto was no longer worthy, replacing it with Ultra Bra[2] and the Beatles. This change to the list is also propagated to the Virtanens’ aunt Maija-page and, naturally, to anyone who has a copy of her list of favorites.

This way the Virtanens can trust that the contact information in the Book of friends is up to date. Mr. Virtanen is also able to pick music according to aunt Maija’s current preferences when she arrives for dinner.

The files related to the use case are in appendix C. The profile on aunt Maija is written in self-explanatory XML, while her home page as well as the page on her from the Virtanens’ Book of friends is written in XHTML. The home page on aunt Maija was written using Microsoft FrontPage with deliberate heavy formatting and sloppy use of structure to reflect the state of current HTML (mis)use.

2.3 Use case 3: Updating an annotated illustration in a document

Mr. Virtanen is currently co-authoring the User’s Guide for the SuperUltra VoiceModem, a next-generation internal voice fax modem, especially suited for the Linux operating system. He has written the chapter on installing the modem, which also includes an illustration of the device. The illustration is drawn by a graphical designer at his company, and was not fully completed when Mr. Virtanen needed the illustration for the chapter. Nevertheless, he used a draft version of the illustration that showed only the essential parts of the modem. He then added some hilights and explanatory text about the modem connectors to the illustration. The original illustration is depicted in figure 2.3, and the the page containing the annotated illustration in figure 2.4.

[2] A popular Finnish band, who are said to have “found the lost sound of the 60’s”

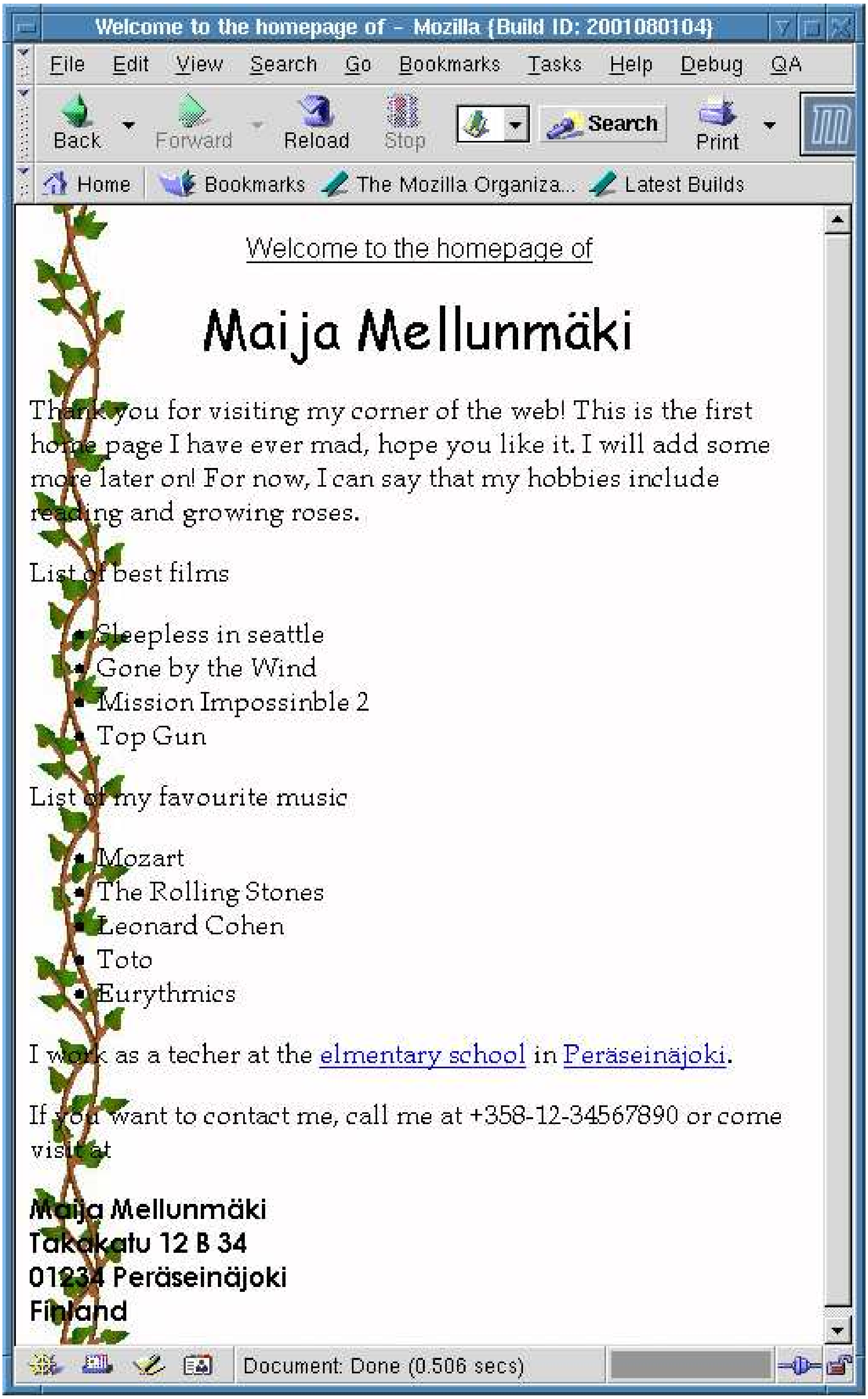


Figure 2.1: Aunt Maija's home page

Figure 2.2: The page on aunt Maija in the Virtanen's Book of friends

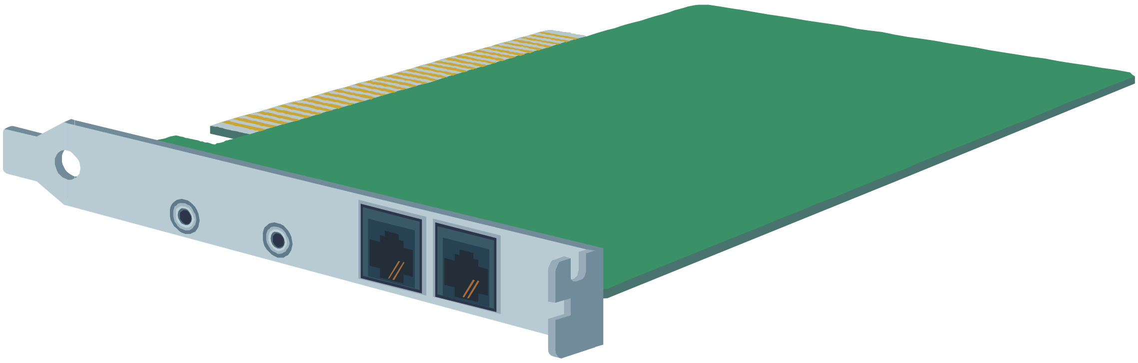


Figure 2.3: Draft version of the illustration of the SuperUltra VoiceModem

Now the graphical designer has completed the drawing, and Mr. Virtanen can update the final version into the manual. The edits Mr. Virtanen made to the draft version are automatically transferred to the final version of the illustration (figure 2.5). Compare this to what has to be done using current tools: the user has to open the final version of the image in an editor and manually add the same modifications he added to the draft version, possibly by copy-pasting parts from the modified draft version. After saving this new version, ha has to manually replace the illustration in the document with the new version.

The fragment of the User's Guide presented in this case is written in XHTML, with an inline graphic in the SVG format. *SVG* (Scalable Vector Graphics) is an XML-based language for describing two-dimensional graphics in XML. SVG allows for three types of graphic objects: vector graphic shapes (e.g. paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. [W3C01a]

At the time of writing, there were to the author's knowledge unfortunately no software capable of correctly[3] displaying inline SVG graphics. The figures have been created by combining separate renderations of the XHTML source and the inline SVG graphics.

The files for the use case not presented here are shown in appendix D.

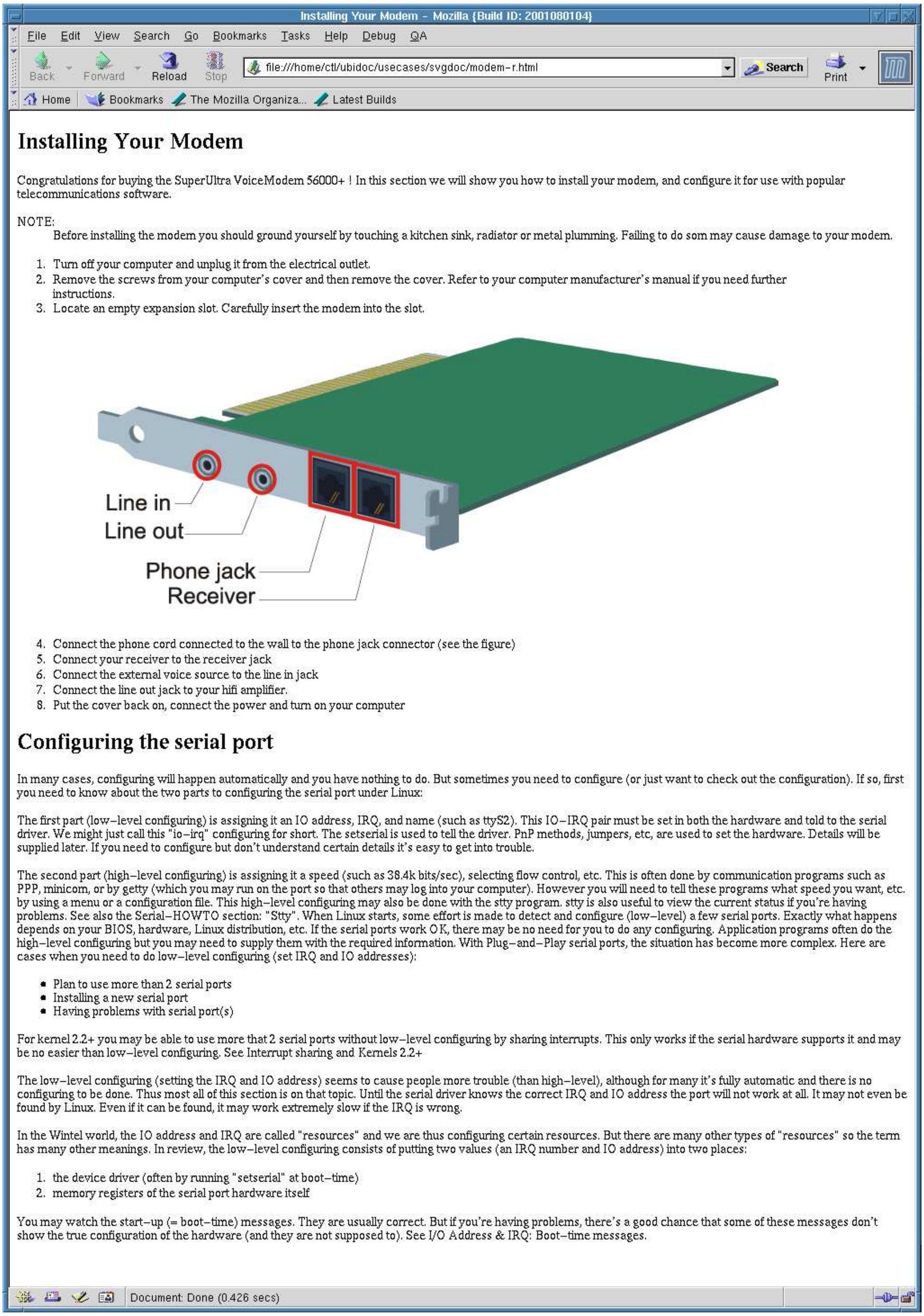
2.4 Use case 4: Merging changes from several authors

Mr Virtanen is also responsible for the finalization of the chapter on troubleshooting for the aforementioned modem. The writing process is in its final stages, and on Friday last week Mr. Virtanen sent the chapter for a quick read-trough to three of his fellow co-workers: Ms. Jaatinen, Mr. Simola and Mr. Ollila.

Mr. Ollila received the mail asking for a read-trough just before he left work on Friday. He spent his weekend out in the Finnish archipelago, but still had some time to briefly look at the chapter on Saturday evening. Initially he had only meant to look at it, but after a while he found himself making some changes.

Neither could Mr. Simola nor Ms. Jaatinen resist making some modifications, once they spotted a few mistakes. Mr. Virtanen was thus quite surprised to receive new revisions from each of them, and is now in the process of combining their changes into a single document.

[3]The Amaya 4.3 [W3C01c] browser by the W3C has some support for inline SVG, but unfortunately did not scale the illustration correctly.



Installing Your Modem - Mozilla {Build ID: 2001080104}

File Edit View Search Go Bookmarks Tasks Help Debug QA

Back Forward Reload Stop file:///home/ctl/ubidoc/usecases/svgdoc/modem-r.html Search Print

Home Bookmarks The Mozilla Organiza... Latest Builds

Installing Your Modem

Congratulations for buying the SuperUltra VoiceModem 56000+ ! In this section we will show you how to install your modem, and configure it for use with popular telecommunications software.

NOTE:
Before installing the modem you should ground yourself by touching a kitchen sink, radiator or metal plumming. Failing to do som may cause damage to your modem.

1. Turn off your computer and unplug it from the electrical outlet.
2. Remove the screws from your computer's cover and then remove the cover. Refer to your computer manufacturer's manual if you need further instructions.
3. Locate an empty expansion slot. Carefully insert the modem into the slot.

4. Connect the phone cord connected to the wall to the phone jack connector (see the figure)
5. Connect your receiver to the receiver jack
6. Connect the external voice source to the line in jack
7. Connect the line out jack to your hifi amplifier.
8. Put the cover back on, connect the power and turn on your computer

Configuring the serial port

In many cases, configuring will happen automatically and you have nothing to do. But sometimes you need to configure (or just want to check out the configuration). If so, first you need to know about the two parts to configuring the serial port under Linux:

The first part (low-level configuring) is assigning it an IO address, IRQ, and name (such as ttyS2). This IO-IRQ pair must be set in both the hardware and told to the serial driver. We might just call this "io-irq" configuring for short. The setserial is used to tell the driver. PnP methods, jumpers, etc, are used to set the hardware. Details will be supplied later. If you need to configure but don't understand certain details it's easy to get into trouble.

The second part (high-level configuring) is assigning it a speed (such as 38.4k bits/sec), selecting flow control, etc. This is often done by communication programs such as PPP, minicom, or by getty (which you may run on the port so that others may log into your computer). However you will need to tell these programs what speed you want, etc. by using a menu or a configuration file. This high-level configuring may also be done with the stty program. stty is also useful to view the current status if you're having problems. See also the Serial-HOWTO section: "Stty". When Linux starts, some effort is made to detect and configure (low-level) a few serial ports. Exactly what happens depends on your BIOS, hardware, Linux distribution, etc. If the serial ports work OK, there may be no need for you to do any configuring. Application programs often do the high-level configuring but you may need to supply them with the required information. With Plug-and-Play serial ports, the situation has become more complex. Here are cases when you need to do low-level configuring (set IRQ and IO addresses):

- Plan to use more than 2 serial ports
- Installing a new serial port
- Having problems with serial port(s)

For kernel 2.2+ you may be able to use more that 2 serial ports without low-level configuring by sharing interrupts. This only works if the serial hardware supports it and may be no easier than low-level configuring. See Interrupt sharing and Kernels 2.2+

The low-level configuring (setting the IRQ and IO address) seems to cause people more trouble (than high-level), although for many it's fully automatic and there is no configuring to be done. Thus most all of this section is on that topic. Until the serial driver knows the correct IRQ and IO address the port will not work at all. It may not even be found by Linux. Even if it can be found, it may work extremely slow if the IRQ is wrong.

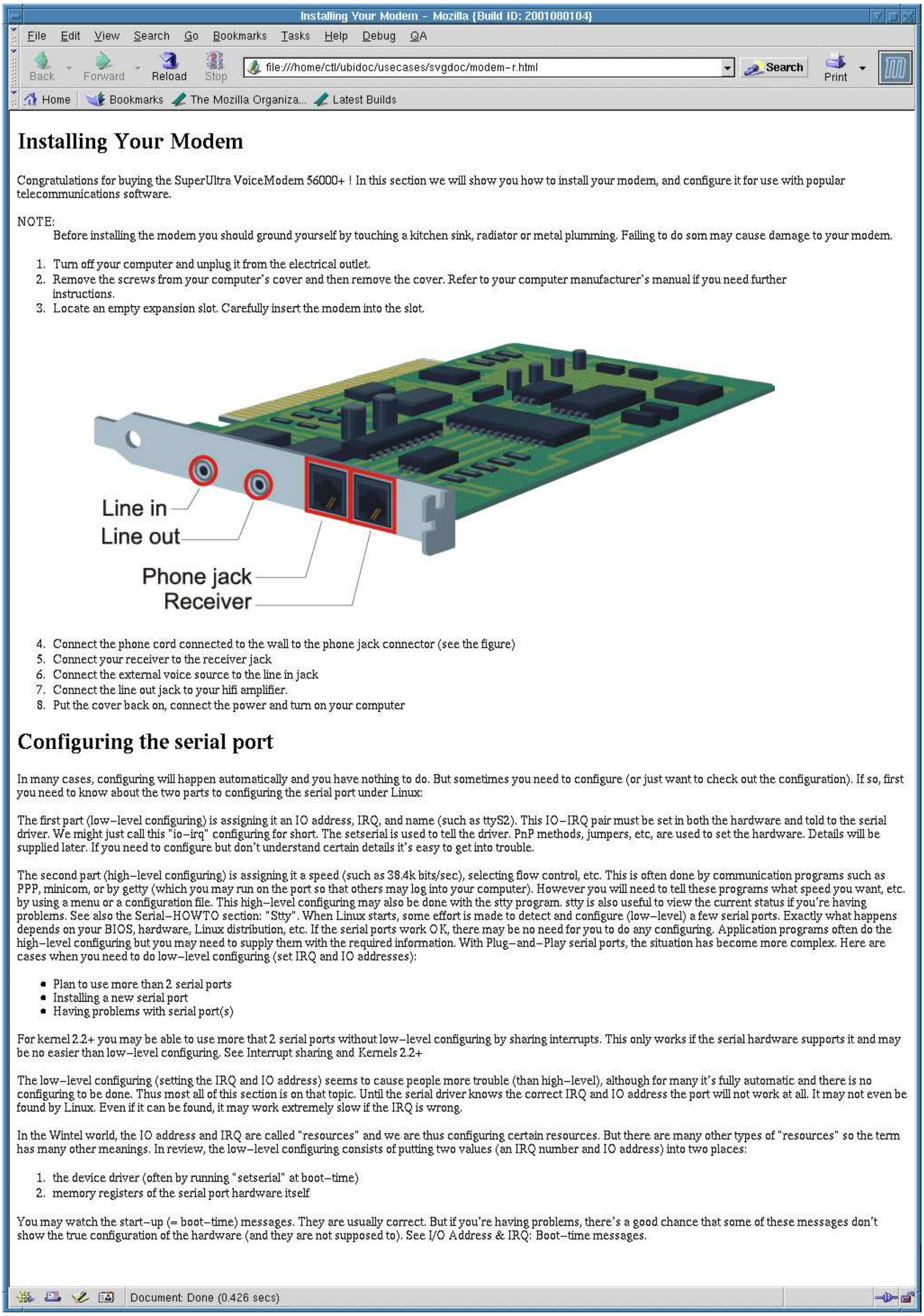
In the Wintel world, the IO address and IRQ are called "resources" and we are thus configuring certain resources. But there are many other types of "resources" so the term has many other meanings. In review, the low-level configuring consists of putting two values (an IRQ number and IO address) into two places:

1. the device driver (often by running "setserial" at boot-time)
2. memory registers of the serial port hardware itself

You may watch the start-up (= boot-time) messages. They are usually correct. But if you're having problems, there's a good chance that some of these messages don't show the true configuration of the hardware (and they are not supposed to). See I/O Address & IRQ: Boot-time messages.

Document: Done (0.426 secs)

Figure 2.4: Page from the user's guide for the SuperUltra VoiceModem



Installing Your Modem

Congratulations for buying the SuperUltra VoiceModem 56000+ ! In this section we will show you how to install your modem, and configure it for use with popular telecommunications software.

NOTE:
Before installing the modem you should ground yourself by touching a kitchen sink, radiator or metal plumming. Failing to do som may cause damage to your modem.

1. Turn off your computer and unplug it from the electrical outlet.
2. Remove the screws from your computer's cover and then remove the cover. Refer to your computer manufacturer's manual if you need further instructions.
3. Locate an empty expansion slot. Carefully insert the modem into the slot.
4. Connect the phone cord connected to the wall to the phone jack connector (see the figure)
5. Connect your receiver to the receiver jack
6. Connect the external voice source to the line in jack
7. Connect the line out jack to your hifi amplifier.
8. Put the cover back on, connect the power and turn on your computer

Configuring the serial port

In many cases, configuring will happen automatically and you have nothing to do. But sometimes you need to configure (or just want to check out the configuration). If so, first you need to know about the two parts to configuring the serial port under Linux:

The first part (low-level configuring) is assigning it an IO address, IRQ, and name (such as ttyS2). This IO-IRQ pair must be set in both the hardware and told to the serial driver. We might just call this "io-irq" configuring for short. The setserial is used to tell the driver. PnP methods, jumpers, etc, are used to set the hardware. Details will be supplied later. If you need to configure but don't understand certain details it's easy to get into trouble.

The second part (high-level configuring) is assigning it a speed (such as 38.4k bits/sec), selecting flow control, etc. This is often done by communication programs such as PPP, minicom, or by getty (which you may run on the port so that others may log into your computer). However you will need to tell these programs what speed you want, etc. by using a menu or a configuration file. This high-level configuring may also be done with the stty program. stty is also useful to view the current status if you're having problems. See also the Serial-HOWTO section: "Stty". When Linux starts, some effort is made to detect and configure (low-level) a few serial ports. Exactly what happens depends on your BIOS, hardware, Linux distribution, etc. If the serial ports work OK, there may be no need for you to do any configuring. Application programs often do the high-level configuring but you may need to supply them with the required information. With Plug-and-Play serial ports, the situation has become more complex. Here are cases when you need to do low-level configuring (set IRQ and IO addresses):

- Plan to use more than 2 serial ports
- Installing a new serial port
- Having problems with serial port(s)

For kernel 2.2+ you may be able to use more that 2 serial ports without low-level configuring by sharing interrupts. This only works if the serial hardware supports it and may be no easier than low-level configuring. See Interrupt sharing and Kernels 2.2+

The low-level configuring (setting the IRQ and IO address) seems to cause people more trouble (than high-level), although for many it's fully automatic and there is no configuring to be done. Thus most all of this section is on that topic. Until the serial driver knows the correct IRQ and IO address the port will not work at all. It may not even be found by Linux. Even if it can be found, it may work extremely slow if the IRQ is wrong.

In the Wintel world, the IO address and IRQ are called "resources" and we are thus configuring certain resources. But there are many other types of "resources" so the term has many other meanings. In review, the low-level configuring consists of putting two values (an IRQ number and IO address) into two places:

1. the device driver (often by running "setserial" at boot-time)
2. memory registers of the serial port hardware itself

You may watch the start-up (= boot-time) messages. They are usually correct. But if you're having problems, there's a good chance that some of these messages don't show the true configuration of the hardware (and they are not supposed to). See I/O Address & IRQ: Boot-time messages.

Figure 2.5: The page from the user's guide with the final illustration

The document merge tool is handy for such tasks. Mr. Virtanen need only select the files that are to be combined (in this case his old version, and the three modified revisions) and the edits made independently by his his co-workers are transferred to his old version. In this case, there are some conflicting edits as well: a chapter modified by Ms. Jaatinen was deleted by Mr. Simola, and the section on the error message “Cannot create lockfile. Sorry” was moved to different places by Mr. Ollila and Mr. Simola.

The merge tool hilights the modified sections, and clearly marks out the conflicting edits, so it is easy for Mr. Virtanen to review the changes and resolve the conflicts.

Although the chapter was non-collaboratively revised by Mr. Virtanen’s co-workers, it was still possible to automatically integrated the fixes into a single document. The possibility of such an *a posteriori* collaboration offered by the document merge tool proved very useful. In this way, the document was reviewed in an ad hoc fashion, without any formal review meeting.

Note how the task of merging the different versions of the chapter exceeds the capabilities of merge tools currently found in word processors such as Microsoft Word from Microsoft [Micro] and StarOffice from Sun [Sun]: documents can only be merged by pairwise comparison, meaning that you have to manually choose either version at the locations that the text differs. A paragraph that is updated as well as moved requires you to manually perform either the move or the update, as with pairwise comparison you have to choose *either* the moved *or* the updated version. Document meta data, such as the definition of styles is not merged at all. Compare this to how the merge tool in the use case merges the updated standard paragraph style in Mr. Ollila’s version to the final document.

The chapter on modem troubleshooting was edited in the word processing application of the OpenOffice office tool suite [OOff]. OpenOffice is an open-source project based on the StarOffice suite by Sun Microsystems [Sun], who released the source code in October 2000. Although currently in pre-alpha stage [OOff01a], the suite is of high interest as it is very capable (features parallel those of the well-know Microsoft Office suite [Sun01a]) and the native file format of the applications is XML-based. At the time of writing, the XML support in the word processing application is mature enough to implement the use case. Both Microsoft and Sun are planning to introduce extensive XML support into their office suites, making them directly usable with the XML merging and differencing tool which will be presented in this thesis.

The OpenOffice suite was preferred over standard XHTML to further demonstrate the interoperability of 3DM with different XML-based languages.

The text on modem troubleshooting is taken[4] from chapter 15 of the Linux Modem-HOWTO [Law01]. The modifications to the text are, of course, only illustratory and do not represent any improvement of the excellent Modem-HOWTO text.

The base version, the new revisions, as well as the desired merged document are in appendix E.

2.5 Use case 5: Maintaining several versions of a web page

When traveling on the bus on his way to work, Mr. Virtanen likes to use his PDA to read the latest news, surf the net, or sometimes write mail. Today he decides to do a brush-up of the WWW page of the chess club he and some friends have formed: the Chaotic Chess Society (CCS). The CCS’

[4]With the author’s consent

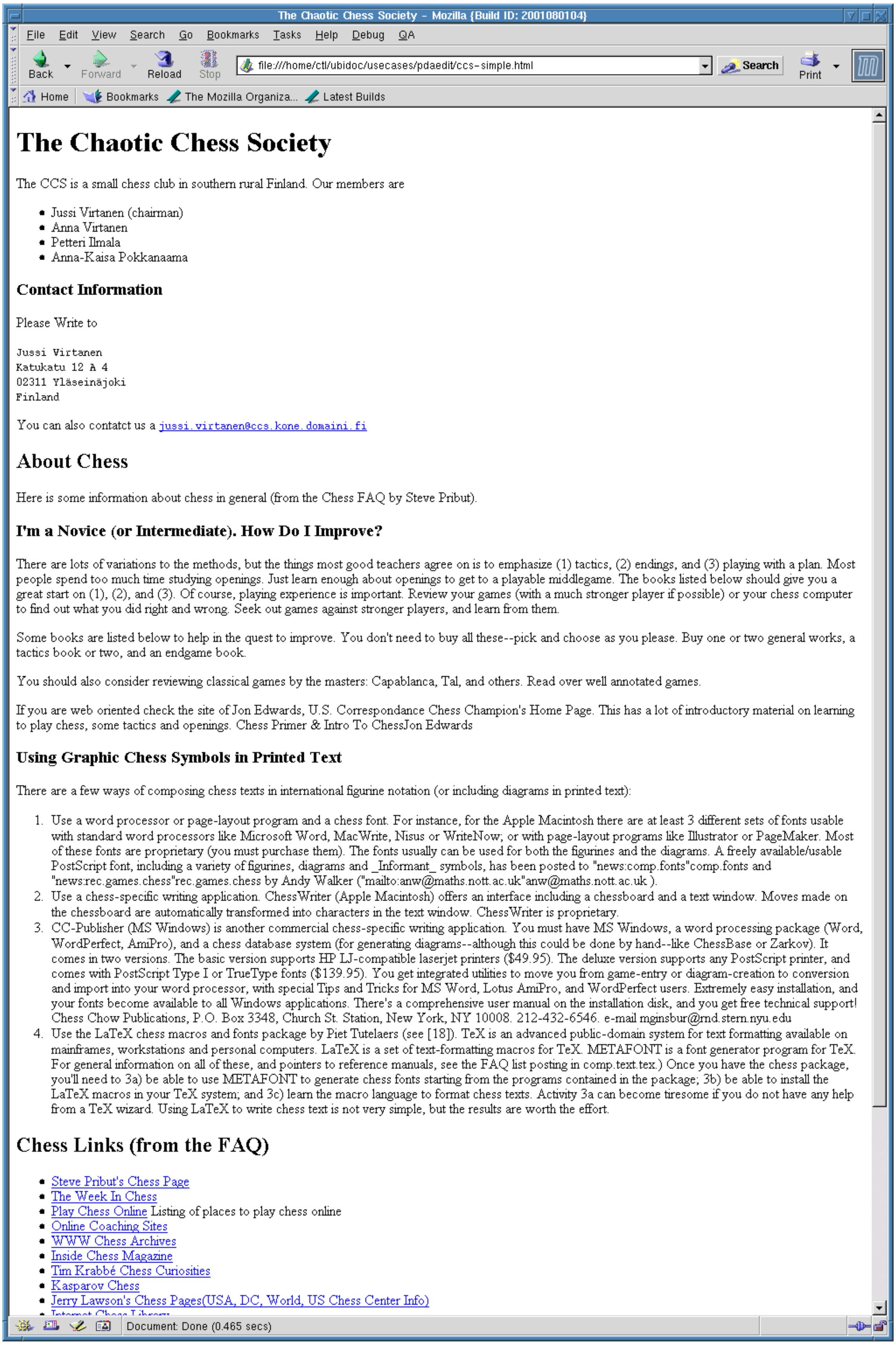
web page is available in two versions: one for full-color large-screen browsers (see figure 2.6) and another, simpler version (figure 2.7), for more limited environments, such as a small screen PDA. The simple layout of the second version also makes the page more accessible for e.g. the visually impaired.

He browses to the simpler version and opens it for editing in his PDA. There are a few errors and corrections to be made: inserting a new hyper link here, remove some extra characters there, and so on.

When done, he saves the revised version. Almost instantly he receives a notification that the changes he made have also successfully been applied to the full version of the page. Mr. Virtanen likes this feature very much: maintaining multiple versions is considerably easier when updates are automatically propagated to all versions.

Both versions of the page are written in standard XHTML. The pages are in appendix F.

Figure 2.6: Full version of the CCS web page



The Chaotic Chess Society - Mozilla {Build ID: 2001080104}

File Edit View Search Go Bookmarks Tasks Help Debug QA

Back Forward Reload Stop file:///home/ctl/ubidoc/usecases/pdaedit/ccs-simple.html Search Print

Home Bookmarks The Mozilla Organiza... Latest Builds

The Chaotic Chess Society

The CCS is a small chess club in southern rural Finland. Our members are

- Jussi Virtanen (chairman)
- Anna Virtanen
- Petteri Ilmala
- Anna-Kaisa Pokkanaama

Contact Information

Please Write to

```
Jussi Virtanen
Katukatu 12 A 4
02311 Yläseinäjoki
Finland
```

You can also contatct us a jussi.virtanen@ccs.kone.domaini.fi

About Chess

Here is some information about chess in general (from the Chess FAQ by Steve Pribut).

I'm a Novice (or Intermediate). How Do I Improve?

There are lots of variations to the methods, but the things most good teachers agree on is to emphasize (1) tactics, (2) endings, and (3) playing with a plan. Most people spend too much time studying openings. Just learn enough about openings to get to a playable middlegame. The books listed below should give you a great start on (1), (2), and (3). Of course, playing experience is important. Review your games (with a much stronger player if possible) or your chess computer to find out what you did right and wrong. Seek out games against stronger players, and learn from them.

Some books are listed below to help in the quest to improve. You don't need to buy all these--pick and choose as you please. Buy one or two general works, a tactics book or two, and an endgame book.

You should also consider reviewing classical games by the masters: Capablanca, Tal, and others. Read over well annotated games.

If you are web oriented check the site of Jon Edwards, U.S. Correspondance Chess Champion's Home Page. This has a lot of introductory material on learning to play chess, some tactics and openings. Chess Primer & Intro To ChessJon Edwards

Using Graphic Chess Symbols in Printed Text

There are a few ways of composing chess texts in international figurine notation (or including diagrams in printed text):

1. Use a word processor or page-layout program and a chess font. For instance, for the Apple Macintosh there are at least 3 different sets of fonts usable with standard word processors like Microsoft Word, MacWrite, Nisus or WriteNow; or with page-layout programs like Illustrator or PageMaker. Most of these fonts are proprietary (you must purchase them). The fonts usually can be used for both the figurines and the diagrams. A freely available/usable PostScript font, including a variety of figurines, diagrams and _Informant_ symbols, has been posted to "news:comp.fonts"comp.fonts and "news:rec.games.chess"rec.games.chess by Andy Walker ("mailto:anw@maths.nott.ac.uk"anw@maths.nott.ac.uk).
2. Use a chess-specific writing application. ChessWriter (Apple Macintosh) offers an interface including a chessboard and a text window. Moves made on the chessboard are automatically transformed into characters in the text window. ChessWriter is proprietary.
3. CC-Publisher (MS Windows) is another commercial chess-specific writing application. You must have MS Windows, a word processing package (Word, WordPerfect, AmiPro), and a chess database system (for generating diagrams--although this could be done by hand--like ChessBase or Zarkov). It comes in two versions. The basic version supports HP LJ-compatible laserjet printers ($49.95). The deluxe version supports any PostScript printer, and comes with PostScript Type I or TrueType fonts ($139.95). You get integrated utilities to move you from game-entry or diagram-creation to conversion and import into your word processor, with special Tips and Tricks for MS Word, Lotus AmiPro, and WordPerfect users. Extremely easy installation, and your fonts become available to all Windows applications. There's a comprehensive user manual on the installation disk, and you get free technical support! Chess Chow Publications, P.O. Box 3348, Church St. Station, New York, NY 10008. 212-432-6546. e-mail mginsbur@rnd.stern.nyu.edu
4. Use the LaTeX chess macros and fonts package by Piet Tutelaers (see [18]). TeX is an advanced public-domain system for text formatting available on mainframes, workstations and personal computers. LaTeX is a set of text-formatting macros for TeX. METAFONT is a font generator program for TeX. For general information on all of these, and pointers to reference manuals, see the FAQ list posting in comp.text.tex.) Once you have the chess package, you'll need to 3a) be able to use METAFONT to generate chess fonts starting from the programs contained in the package; 3b) be able to install the LaTeX macros in your TeX system; and 3c) learn the macro language to format chess texts. Activity 3a can become tiresome if you do not have any help from a TeX wizard. Using LaTeX to write chess text is not very simple, but the results are worth the effort.

Chess Links (from the FAQ)

- Steve Pribut's Chess Page
- The Week In Chess
- Play Chess Online Listing of places to play chess online
- Online Coaching Sites
- WWW Chess Archives
- Inside Chess Magazine
- Tim Krabbé Chess Curiosities
- Kasparov Chess
- Jerry Lawson's Chess Pages(USA, DC, World, US Chess Center Info)

Document: Done (0.465 secs)

Figure 2.7: Simplified version of the CCS web page

Chapter 3

The Structural 3-way Merge Problem

The smooth synchronization of data portrayed in the use cases is not reality today. Clearly, data synchronization can be made much more transparent and flexible, and it is in the context of this vision that the research problems of this thesis are stated.

As background we introduce the concepts of data synchronization and a ubiquitous computing environment, both with an emphasis on mobility. Against this background we then state the research problem of the thesis, which is to design a structural merge tool that fulfills the set of requirements we derive from the use cases and the ubiquitous, mobile environment it is intended to operate in.

3.1 Data synchronization

Data synchronization (see e.g. [SML01b]) is traditionally understood to be the process of making two sets of data look identical. An exact definition of the term seems to be lacking (the term is not included common computing dictionaries), and it is used in slightly different meanings depending on the context. Here, the term will have the following meaning:

Data synchronization Assume two sets of data that have some parts in common. Data synchronization between the sets is the process of making the common parts identical, after changes have occurred in either or both sets. The synchronization process should not ignore changes made to the common part in either set.

As defined here, data synchronization is also strongly related to the concepts of *data reconciliation* (e.g. [MD94]) and *data integration* (e.g. [Cha99, pp. 23-25]).

As an example of data synchronization, as defined above, consider two data sets $S_1 = \{a, b, c\}$ and $S_2 = \{b, c, d\}$. Now, if S_2 is updated to $\{b, c', d'\}$ and synchronized with S_1, the update to $c \in S_1 \cap S_2$ should be propagated to S_1, which when synchronized becomes $\{a, b, c'\}$.

The need for data synchronization frequently arises in everyday life. A typical scenario is having multiple copies of the same data (or a part of it), all or some of which are updated asynchronously as exemplified by use cases 2 and 4. Another is having the same data on your main PC and a PDA, in which case you need to synchronize between the PC and the PDA (e.g. use cases 1 and 5).

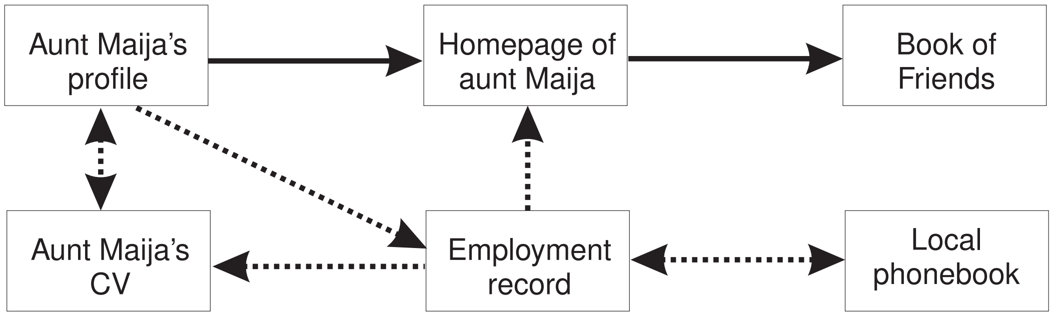


Figure 3.1: Chains of derived data.

Data synchronization is supported by practically all PDAs (such as those from [Palm], [Psion], and those based on the Windows CE operating system from [Micro]) and groupware tools (e.g. Lotus Notes [Lot] and Novell Groupware [Nov]). Here we are particularly interested in cases where the data to be synchronized is located in mobile and wireless devices, including PDAs, where different branches of a piece of data may be created during periods of decoupling or weak coupling.

Pieces of data are not just related pairwise; they usually form chains, where each piece of data is derived from one, or several other pieces. For instance, consider use case 2, in which aunt Maija's home page contains some information from her profile. The Virtanens, in turn, have included some of this information into a document of their own. Updating the profile should not only cause the home page to be updated, but the document at the Virtanens should be updated as well. Figure 3.1 illustrates such synchronization chains. The solid arrows show the profile-home page-book of friends chain present in the use case, and the dashed arrows show some more possible synchronization chains among other related documents not presented in the use case. In the thesis we strive to handle synchronization without imposing any restrictions on how the data may be related.

Data synchronization has been classified in [CP00],[CHKS94] and [MD94]. The aspects of particular interest to this thesis are listed below.

- **Nature of the data synchronized** [CP00]

  We define two (non-exhaustive) categories, *linear* and *structural*. Linear data consists of an ordered list of items (such as lines in a text document), whereas structural data is an ordered tree (e.g. an XHTML document).

- **Synchronization Granularity** [CP00]

  The granularity defines the smallest piece of data, or *atomic data*, handled by the synchronization. After synchronization a piece of atomic data is always identical to its counterpart in the other data set. As an example, if the granularity is "file" when synchronizing a set of HTML files, each file is treated as an atomic object. If the granularity is "HTML element", the synchronization process treats the nodes in the HTML files as atomic units. If two changed HTML files were to be synchronized, the former approach would only be able to present the user with the choice of which file to use, while the latter approach could generate a merged HTML file.

- **Data locking** [MD94]

Data locking schemes may be used to control read and write access to data (see [Gol99, Tic85]). The most well-known type of locking is exclusive write locking, which is used to guarantee that only one user is allowed to modify a piece of data at any given time, thus preventing update conflicts.

Analogous to synchronization granularity, we can also define different locking granularities (i.e. the smallest piece of data that can be locked). Synchronization that does not relay on locking is said to be *optimistic* ([BSV98, Ask94]). With optimistic synchronization, anyone may modify the data and subsequently synchronize it with other updates, at which point any conflicts that arise are reconciled.

- **Coupling** [MD94, BSV98]

  The data sets to synchronize may exhibit different levels of coupling, meaning the availability and capacity of information channels between the data sets. When data sets are connected by a high capacity channel of high availability, we say that the sets are tightly coupled, and when the channel is of low capacity and low availability we say that the sets are weakly coupled. Typically, data sets physically residing in the same computer, or in computers connected by a LAN are tightly coupled, while data sets on wireless devices are typically weakly coupled, including occasionally being *decoupled*, during periods when the network is unavailable.

- **Synchronization topology** [CP00]

  This concept defines restrictions on which other data sets a data set may be synchronized with. The strictest variant is the dedicated-pair topology, in which a data set may only be synchronized with exactly one other data set. A completely unconstrained topology is called peer-to-peer, in which any data set may synchronize with any other set.

  As an example, consider the shopping lists in use case 1. One synchronization topology may restrict the versions in Liisa and Mr. Virtanen's PDAs to be synchronized only to the common shopping list, whereas a peer-to-peer topology would also allow Mr. Virtanen's PDA to synchronize directly with the shopping list in Liisa's PDA.

- **Differential transfers**

  Some synchronization methods need to transfer the data sets in their entirety over the network in order to perform synchronization, while other methods are able to send only the difference between the old and new versions. Weakly coupled data sets benefit from differential transfers, as limited network capacity is potentially saved.

- **Application support**

  Some synchronization methods require explicit support for synchronization from the applications producing the data sets, such as the presence of unique object IDs, timestamps or edit histories, whereas other methods require no such meta information to be included in the data, and thus require no application support.

3.2 Ubiquitous computing in the UbiServices project

The term *ubiquitous computing* (or ubicomp for short), introduced by Mark Weiser in [Wei91] describes a world where the use of computers have moved from the conscious to the unconscious.

Weiser compares the ubiquity of computing to writing: in early history writing was an art practiced by few, requiring knowledge of ink-making, production of the media (such as clay) and pens, whereas today writing is ubiquitous: it is all around us, and used whenever we want to convey information. In the same way, the computers of the 21^{st} century should become ubiquitous, allowing us to focus on the task at hand and not on the tool, the computer.

Weiser states several characteristics of the ubiquitous computing environment in [Wei93]:

The number of computing devices will be orders of magnitude larger than today, moving past the paradigm of personal computers and digital assistants towards computers as physical objects, as common as books or sheets of paper. With this follows the embodiment of information: an electronic document is no longer an abstract entity inside a file system, but rather a concrete object, such as an electronic "pad".

A fundamental property of the ubiquitous devices is their interconnectedness and awareness of location. A computer that knows its location as well as the devices in its immediate neighborhood, can offer a range of services in an intuitively pleasing way. Physical closeness is fundamental to human perception — if I desire to synchronize the contents of a pad with another, I could simply place them on top of each other (or something similar), the idea being that if the pads are physically close they should be connected in some way.

In the UbiServices project we share the basic notion of Weiser to move the act of "using a computer" towards the unconscious. One of the goals of the three-year project, of which the first is in progress, is that a vision of computing in the next decade gradually should emerge. Does this vision include Weiser's embodiment of information [Wei91] or perhaps the use of ambient media (such as background lighting) to "display" information, as envisioned in [IU97]? We do not yet know.

We refer to the various ways in which assistance is provided by computers as *services*. A service may be bound to device (e.g. the "telephony" service of a mobile phone) or something which can be "present" in almost any device, such as the world wide web.

Our current research on services is focused on the following aspects related to ubiquitous computing, which in our opinion presents interesting and relevant research topics and forms a good starting point, from which our vision can emerge:

- **Context Awareness** [DA00]

  A context aware service utilizes contexts (e.g. what is the current location, what people are present, what is the nature of the gathering etc.) to enhance its operation. An example of a context aware service is an application that shows the schedule of the bus stop closest to the user, or a mobile phone that does not ring in the middle of an important meeting.

  In the UbiServices project, context awareness plays a central role, including the implementation of support for context awareness, as will be described in upcoming work.

- **Location as an enhancement, not a limitation**

  Services should not be unnecessarily limited to a particular location. If I want to pay a bill from my mobile phone, I should be able to do it, and not have to log on to my desktop computer. Still, services should be aware of the location. I should, for instance, be able to query a dining service about the "nearest restaurant". This aspect is strongly related to context awareness and mobility.

- **Service discovery**

Service discovery deals with the problem of discovering services available from the user's as well as other services' point of view. A related question is how to present services to the user and other services in terms of user interfaces and programming APIs. Service discovery in an ubicomp environment poses some additional challenges; in fact one could say that service discovery in a ubicomp environment is an oxymoron: If the services are on the edge to the unconscious, how can they be discovered? Is a service, that we have to discover, in fact not ubiquitous at all?

- **Universal data access** [Abo96, BHL01]

  Today, far too much of our time is spent trying to make different programs talk to each other, having to convert and move data around in order to edit it, make it available and to keep it up-to-date. We desperately need services that transparently integrate with another and automatically understand each others' data.

  The current situation can be illustrated with something as simple as publishing a document on the web. If it was written with Microsoft Word, you have to convert it to HTML and copy it to a directory visible to the web server. The published document is essentially read-only, so if somebody makes a change they have to make a copy of their own, edit it and send it to you. Of course, they edited the HTML version, and your original version is in Microsoft Word, meaning that yet another conversion, with possible loss of formatting, has to take place.

- **Mobility** [Sat96, Dea98]

  With mobility we understand mobility of the physical device as well as that of services, including the constraints a mobile service must adhere to. Technologies for enabling physical mobility of devices include various wireless telephony network standards (GSM, UMTS), the 802.11b wireless LAN specification and Bluetooth. Various technologies for service mobility also exist, ranging from mobile user interfaces (the World Wide Web) to code mobility (the Java programming language from [Sun]).

  Strongly related to mobility is the ability of services to adapt when moving form one device to another. For instance, an online banking service need to adapt its screen usage, depending on if it is used from a PC or a mobile phone.

- **The peer-to-peer computing model**

  Traditionally, services have been implemented in a client/server fashion, the WWW being perhaps the most widespread example. In the peer-to-peer model, we envision that every device can have server functionality, eliminating the need for centralized servers and providing increased flexibility. Well known peer-to-peer applications include the usenet [Use] and the Napster file sharing service [Nap].

- **Interaction between services** [BHL01]

  In order to reach their full potential, services must also be aware of other services available, and be able to interact — usually in an ad-hoc manner. For instance, when booking a flight ticket, the booking service should also let you pay for the ticket, and perhaps take a travel insurance, requiring interaction between the ticketing, online banking and travel insurance services.

What does the technological environment, in which the services are implemented in, look like? Predicting the evolution of technology is no easy task, but some reasonable assumptions can be made.

- Processing power per watt will continue to increase. This means small devices will be faster and/or consume less power.
- The capacity of wired networks will continue to grow, whereas the capacity of wireless networks will still be fairly limited. Additionally, wireless devices will occasionally be without a network connection, due to e.g. network usage costs and radio interference.

Especially mobile devices suffer from the limitations of wireless networks. As we consider mobility to be one of the fundamental aspects of the ubicomp environment, it means we have to engineer services for a weakly coupled environment. It is also in the context of mobile devices that the peer-to-peer computing model becomes important.

Consider two persons meeting at some place where network availability is limited, as for instance the Finnish archipelago. The persons need to collaborate using their PDAs. If the PDAs are client/server-based, this is either not possible at all or slow and expensive as the "home" servers of the PDAs are not easily reachable. If the PDAs, and the services they provide, are capable of autonomous operation and peer-to-peer ad-hoc connectivity, no such problems arise.

Moving data away from the clients onto centralized servers, with increased access latencies and costs, makes little sense in a mobile world. This situation is not likely to change, either.

3.3 Thesis problem statement

In this thesis we will look at data synchronization with an emphasis on, but not limited to, ubiquitous computing. In the UbiServices project we feel that data synchronization is one of the basic functionalities that should be available in a ubiquitous computing environment.

The uses cases presented in chapter 2 demonstrate the need for data synchronization in several everyday situations. In a ubicomp world we can easily imagine that data synchronization is even more important. In fact, we propose that it plays an important role in realizing the goal of universal data access, by relieving the user and services of the burden of manually having to keep various pieces of related data up-to-date across different formats.

We will now state the general research question of this thesis, and then proceed by focusing on the aspects of the question that will be addressed in the thesis.

> *How can we, with minimal application support, synchronize related structural data in a weakly coupled environment?*

We state that application support should be minimal, because we do not believe that there will be an all-encompassing standardized method of achieving synchronization, such as a central data repository to which all applications would connect. We should be able to support any application, as long as the structure of its data is open, i.e. not encoded in an (partially) unknown manner. For instance, the structure of any XML document is open and accessible, whereas the structure of many proprietary file formats is unknown, and therefore impossible to access.

By related data we mean data sets that have some parts in common, as exemplified in section 3.1. The ability to handle structural data (i.e. any ordered tree) in a meaningful way presents a challenge that goes beyond existing text merging tools. We focus on a weakly coupled (or wireless) environment, as this is the environment in which we envision that ubicomp services will most likely operate in (as stated in section 3.2).

How can we know which parts the data structures have in common? One approach is to maintain some kind of meta information for the data structure, which contains this information (e.g. "the subtree rooted at node x in tree A is the same as the subtree rooted at node y in tree B"). Another approach is to automatically identify the common parts. In this thesis we will consider the latter approach, as the former would require application support to generate the meta information.

Consider the related ordered trees T_1 and T_B. Assume T_B changes to T_2, and we want to propagate these changes to the parts of T_B present in T_1. We will of course need T_1 as well as T_2 to accomplish this. We will also need T_B, in order to identify the common parts of T_1 and T_B. This task is known as a 3-way merge. To emphasize that the nature of the merged data is structural, i.e. an ordered tree, we talk about a structural 3-way merge:

Structural 3-way merge Assume T_1 and T_2 are ordered trees derived from the tree T_B. The 3-way merge of the trees T_1, T_B and T_2 is an ordered tree T_M, where the changes between T_B and T_1, as well as the changes between T_B and T_2 are incorporated. The tree T_B is called the base and the trees T_1 and T_2 are branches.

By using a 3-way merge instead of the more traditional diff/patch approach we avoid the *persistent naming problem*. If we were to generate the diff $T_B \ominus T_1$ and apply it to T_2, we would need some means of identifying the nodes in T_2 that the edit operations in the diff should be applied to. This is not as easy as it seems, since the structure of T_2 may differ radically from T_B (meaning that, for instance, paths cannot be used to identify nodes), and we have not assumed the presence of any node IDs. In other words, we have no persistent name for the parts of T_2 we wish to apply the changes to.The solution for text differencing is to include some of the additional data in the diff as context for the edit operations, and it seems we would have to define some sort of "tree context" as well. Presumably, this tree context would not allow edit operations very close to one another, just as this is not possible with the "context diffs" that the UNIX `diff` tool produces.

We notice that a structural 3-way merge will have no trouble working in a weakly coupled environment, as the trees T_1 and T_2 may be totally decoupled up to the point when the 3-way merge occurs.

A structural 3-way merge tool will thus allow us to perform synchronization of related structural data in a weakly coupled environment, with minimal application support. The main task of this thesis will therefore be to design and prototype such a tool. In addition, we would like to be able to produce differences between trees, in order to enable differential transfer of data, and thus save bandwidth.

In the thesis, we will *not consider* how the data to be synchronized is moved over the network or user interface aspects. Nor will we define high-level merge policies (e.g. under what circumstances the tool should be run). We will strictly focus on the design and prototyping of the algorithms underlying structural 3-way merging and differencing. Nonetheless, a description of the environment, and the properties of that environment, in which the tool will operate is essential when determining the requirements for the merge tool, and have therefore been included.

The structural 3-way merge tool should also be able to detect and express any conflicts that occur during merging. In addition it should be able to show, in the merged tree, the changes that have been made with respect to the base tree, a feature that will be useful when comparing documents (as in use case 1) and when verifying that the merged data is acceptable. For instance, assume that a 3-way merge is performed on text documents. In order to make sure that the merged document is meaningful and consistent it has to be reviewed by a human. Having the changed parts clearly marked out significantly aids the task of the human reviewer.

The notion of a 3-way structural merge tool is a very powerful one, indeed:

- The user is no longer limited to exact synchronization. The data may contain changes of his own (which may be huge compared to the original data, e.g. considering this thesis to be a modification of one of its figures), and he is still able to synchronize with an updated version, *preserving his own changes*. In the use cases, the vast majority of the synchronization tasks are of this type.

  For instance, consider the update of the suggested shopping list in use case 1 (step 5 in table 2.1 on page 10): the suggested list contains modifications with respect to the common list, and these modifications are preserved when the suggested list is synchronized. The update of the draft modem drawing in use case 3 is another example. Here, Mr. Virtanen has made some modifications to the draft version, and the modifications are preserved when updating to the final version of the drawing.

- The user is free to modify the data at any time, no locking schemes nor tight coupling (i.e. a persistent network connection) is needed. When the network becomes available, the user can merge his data with any other updates.

  Synchronization in a weakly coupled environment, where no locks exist and data may be edited freely at any time is particularly evident in use cases 1, 4 and 5.

We can now state the specific research problem of this thesis:

Research problem Assume T_1 and T_2 are two ordered trees derived from the tree T_B. The research problem of this thesis is to design and prototype a tool that can

1. Perform a structural 3-way merge of the trees T_1, T_2 and T_B and detect and express any conflicts that occur during merge. This is the *tree merge problem*.
2. Produce the difference between two trees T_1 and T_2 and apply this difference to T_1 to obtain T_2. These problems are the *tree differencing problem* and the *tree patching problem*.
3. Produce a trace of the 3-way merge that expresses the changes, and their origin, in the merged tree compared to the base tree.

In addition, we consider the following hypothesis:

Research hypothesis Provided the structure of the trees T_1, T_2 and T_B reflects the structure of the data they represent, it is in most cases possible to create a semantically valid structural 3-way merge without the merge tool having knowledge of the semantics of the trees.

The statement in the definition that the structure of the trees should reflect that of the data they represent is perhaps best explained by a few examples.

Consider a text, consisting of chapters, sections, subsections and body text. A natural, intuitive way is to encode this as a tree with a root node, beneath which we have chapter nodes. The chapter nodes have section nodes as children, who in turn have subsection nodes, and finally text nodes

```
<thesis>
  <title>The 3DM tool</title>
  <chapter>Introduction</chapter>
   <p>The 3DM ...</p>
  </section>
</thesis>
```

```
<thesis>
  <text>The 3DM tool</text>
  <chapter start=1 end=314159/>
  <section start=32 end=2178/>
  <p start=111 end=222/>
</thesis>
```

Figure 3.2: Different XML encodings of a text. The encoding to the left reflects the logical structure of the text, whereas the encoding on the right does not.

(e.g. the [OAS01] format). An considerably less meaningful encoding would be to have say, a single node beneath the root containing all the text, with additional children of the root containing the positions in the text that belong to chapters, sections, subsections etc. See figure 3.2.

As another example, consider a drawing, consisting of several layers and compounded objects that have been rotated and scaled. In a well-formed tree encoding of the drawing this structure of object grouping, scalings and rotations is preserved, whereas a less well-formed approach would be to list all the paths (along with fill attributes etc.) in the drawing as the children of the root, with no additional structure.

A tree whose structure reflects that of the data is said to be well-formed.

Expressing data hierarchically inside a computer program is a very fundamental aspect of computer science. The hierarchies used inside a program very often map directly to the logical hierarchies of the data processed by the program. The well-formedness criteria states that this "natural" hierarchy should also be visible in the tree structures accessible to the 3-way structural merging tool.

It is commonly believed that semantically valid structural merges cannot be performed without the merge tool knowing the semantics of the data it is merging. As the research hypothesis states, the author feels that it is in fact possible to perform such merges without the knowledge of the semantics. We will not research the hypothesis in detail in this thesis, but rather try draw some preliminary conclusions from the results of performing merges on the data provided in the use cases.

3.4 Merge tool requirements

In this section we present the high-level requirements on the merge tool, based on the use cases and the environment of the tool. These requirements will be further refined and implemented in chapter 5 and section 5.4.

As we stated previously, we want the tool to be able to operate on data generated by almost any application (as exemplified by the use cases). On the other hand, most commercial applications of today use proprietary file formats, which do not expose their internal structure. We seek to find a format that will be widely supported, exposes the structure of the data, and is sufficiently standardized to allow us to write a tool that can process any data adhering to the format. XML is such a format that is rapidly gaining ground. Due to its simplistic and self-explanatory nature [W3C01e], XML is also a significant step in the direction of application-independent and human legible data. XML is thus a natural choice for the merge tool. The tool should, however, be extensible with other parsers, for e.g. HTML or LaTeX.

Another important requirement is that the operation of the tool should be easy to understand, both in principle and in practice. A tool that makes unexpected and erroneous changes in the merged data that do not match those made in the input of the merge tool is not desirable. All moves, copies, deletes and updates should correctly be reflected in the merged output.

An example of unexpected behavior when synchronizing is if a change in the first branch to a part that is absent in the second branch would cause any changes to the second branch. The requirement will be further elaborated in section 5.3.

A particularly important situation that occurs frequently in the use cases is that both branches have been structurally edited with respect to the base, i.e. the original structure (as opposed to only the content of the tree nodes) of the base is modified in both branches. For instance, in use case 3 we can consider the draft version of the illustration to be the base. The first branch is then the users' guide, in which a significant amount of structure has been inserted around the original illustration. The second branch, which is the finished illustration, also has structural modifications with respect to the base in the form of added drawing elements (SVG paths).

On the tree node level structural editing in both branches corresponds to nodes being moved in one branch and their content as well as child lists being updated (due to inserts and deletes of new nodes) in the other.[1] Hence, we have named the case of mutual structural editing the *move/update case*.

Moving onwards to architectural requirements we want the merge tool to be able to support decoupled and decentralized peer-to-peer operation, as these are aspects of mobile ubicomp that we feel are important (as stated in section 3.2). Since we want the tool to be able to access the structure of the data, we set the granularity of synchronization to the XML node level, treating text nodes and the names and attribute/value pairs of element nodes as atoms. A 3-way merge tool imposes no restrictions on synchronization topology. Since the peers are assumed to be weakly coupled we use an optimistic synchronization approach that does not use any locks (in case locks were used, one could not start editing a document during decoupling, as locking cannot be performed in a decoupled state).

As already mentioned we should be able to generate diffs, which we want to be reasonably compact. Current text differencing tools do quite a good job on XML, except in the cases where there are changes in the textual representation that do not reflect any changes in the tree structure (e.g. changes in whitespace) and when subtrees are moved or copied, in which case text differencing tools generate a sequence of line inserts and deletes. Ideally, our differencing algorithm is "tree aware", and provides support for subtree moves and copies in the differencing algorithm.

Although we can assume a fair amount of computing power in the peers, the merge tool should also be reasonably efficient — with a time and space complexity preferably less than $O(n^2)$, where n is the combined size of the inputs and outputs of the tool. The architecture should also be modular, allowing for easy improvement and modification of the tool.

[1] Note that nodes that are moved (relative to their parent) in both branches usually represent a conflict, as the destination of the move becomes ambiguous.

Chapter 4

Work on Synchronization, Merging and Differencing

In this chapter we will make an overview of data synchronization approaches used in existing applications, as well as present work related to the structural merging and differencing.

In section 4.1, we begin with the overview of synchronization approaches currently used, in order to survey if there currently exists any applications for structural 3-way merging, find out what their limitations are, and to get ideas for the design of 3DM. The overview consists of examples of current applications, open source as well as proprietary, and technologies representing different synchronization approaches.

Work related to 3DM in also include tools for merging and differencing data, which are presented in sections 4.2 and 4.3. Finally, in section 4.4 we look at algorithmic research related to 3DM: algorithms for data merging and for producing diffs.

4.1 Overview of synchronization approaches

CVS - the Concurrent Versions System

The Concurrent Versions System (CVS) [Berl90, Ced93] is a widely used document version control system. The architecture is client/server based with a central repository for the documents. The users check out local copies of documents from a repository, make changes, and commit the updated versions to the repository. Typically, check-outs in CVS are non-exclusive, meaning that multiple users may have a document checked out for editing simultaneously.

As concurrent editing of a file is allowed, the need for merging changes sometimes arises. Typically, users A and B have checked out a file, they both modify the it, and A commits the changes to the repository. Now, when B tries to commit his changes, CVS informs that the changes made by A need to be merged into his version before he may commit.

CVS is able to automatically handle the merging quite well provided the document is a line-based plain text file. Naturally, CVS cannot merge arbitrary binary files, and there is no particular support for merging structured data, except as plain text files (in which case the structure is not acknowledged).

CVS can be characterized as being a tool capable of plain text merge, not reliant on edit history nor on a strict locking scheme (files need not be locked before they are edited) and capable of decoupled operation, meaning that you have a local copy that you can edit at any time and that the repository only needs to be accessed when explicitly checking in new changes or getting the latest updates. Merging works on any type of plain text file, and does not require any special application support.

We shall see that the CVS model of merging and architecture is quite suitable for the environment 3DM is designed to work in. CVS has provided much inspiration for the design of the architecture of 3DM: the lack of strict locking, operating on files directly and not requiring applications to support a particular synchronization application interface, using local copies of files and not requiring a tightly coupled network are all aspects we will find useful when designing 3DM.

Groove

Groove by Groove Networks [Gro] is a peer-to-peer collaboration tool that allows users to set up common shared spaces. The shared space supports collaboration on such activities as text document writing and drawing, instantly synchronizing any changes to all participants.

Groove is also able to work in an off line state and synchronize the shared space when the network becomes available again. This is accomplished by keeping a local history of all changes made and using a "relay service". The relay service is always online and records all edits made to the shared space since its peer went off line. When the peer is reconnected, it transmits the local edits to all other members of the shared space and receives their edits from the relay service, thus accomplishing a merge of the shared spaces. Documents created outside Groove can be shared, but can not be merged in this way. [Gro01a, Gro01b]

Groove is thus able to perform structural merge of data by remembering edit histories. It appears[1] that each Groove tool is responsible for interpreting the edit commands (and thus the edit history), and thus the merging is specific for that tool.

At the time of writing Groove is still in beta testing.

Lotus Notes and Novell Groupwise

Lotus Notes by Lotus Development Corp. [Lot] and Groupwise by Novell [Nov] are two widespread examples of collaborative software, or "groupware", providing such functionalities as sharing and version control of documents, messaging and scheduling. Both are client/server solutions, meaning that the data is kept on centralized servers as much as possible.

Both products incorporate support for the mobile user by allowing offline work. Offline support is not, however, as transparent as in Groove. Synchronization is achieved by replication, meaning that the most recent version of a piece of data is copied to all servers. Notes replicates data on a field-by field basis, meaning that hierarchical data structures created inside Notes can be merged to some level. Groupwise's approach is to use the common check-in/check-out paradigm for documents as well as a queue of outstanding operations (such as edits to an item). The queue is emptied to the server once the client is reconnected. [Lot00, Nov98].

SyncML

SyncML [SML01a] is an XML-based data synchronization protocol pushed by the SyncML Consortium [SML], whose founding members include IBM, Lotus, Nokia, Ericsson and Motorola. The

[1] Technical details about Groove are not publicly available.

protocol is an attempt to standardize the way data synchronization messages are exchanged between devices. SyncML support is expected to start appearing in a wide range of handheld devices, such as PDAs and smart mobile phones. The basic architecture is a client/server configuration, where the server is constantly online, and the clients are off line for most of the time.

The client and server keep an edit history of all modifications made since the last time of synchronization (or initial upload of the data). Synchronization is accomplished by having the client transmit the edit history to the server, which incorporates the changes into its version and at the same time resolves any conflicts. To complete the synchronization, the server then sends its edit history back to the client.

The SyncML protocol specification does not define how merging of the edit histories is accomplished: that task is the responsibility of the SyncML-enabled application. The protocol does not specify any mechanism for locking data. [SML01a]

WebDAV

WebDAV (also known as just DAV), defined in [Gol99] is an extension to HTTP that adds distributed authoring and versioning capabilities to the protocol. The purpose of WebDAV is to allow users to perform remote web authoring operations including creation, editing, copying, moving and deleting of web resources. A web resource is typically a file or collection of files identified by an HTTP URL. The computing model of WebDAV is client/server.

WebDAV includes locking of web resources. Both exclusive locks, which prevent users from simultaneously editing a resource and shared locks, which restricts simultaneous editing to those users sharing the lock, are supported. In the case of shared locking, a need for merge of the updated resources may arise. WebDAV does not define how the merge should be accomplished, though. [Gol99]

There are no explicit features for decoupled operation in the protocol — off line editing is accomplished by locking the resource, modifying a local copy and uploading the modified copy at a later instant.

Version control and configuration management of web resources was originally in scope of the IETF WebDAV Working Group but is now handled by the DeltaV working group [DeltaV]. The objective of the working group is to define extensions to the HTTP and WebDAV protocols that will provide support for version control and configuration management. The DeltaV working group mentions differencing and merging as being in scope of their work, but only on a protocol level.

4.2 Differencing tools

Differencing tools are relevant to the development of 3DM on two accounts: the first (and obvious) is that 3DM should be able to produce the difference, or delta, between two XML documents in order to save the amount of data that needs to be transmitted or stored. The second is that differences can be thought of as edit histories, and by combining these edit histories we can presumably perform a 3-way merge.

In the following sections we will take a look at existing differencing tools and the diffs they produce.

UNIX diff

The well-known UNIX differencing tool `diff` is capable of producing the line-by-line difference between two text files. Based on the algorithm by [Mye86], it produces the minimum-length edit script to transform the first input file to the second using line insert and delete operations. Diff does not detect line moves; these are treated as delete/insert pairs.

The tool is not directly applicable to structural differencing, as it is commonly believed that general structural differencing problems cannot be reduced to linear differencing problems [ZS97].

TopBlend

TopBlend [CDH00] is a differencing tool for HTML files, based on earlier work presented in [Berk96]. TopBlend is able to show the differences between two HTML files in either as a side-by-side view or as combined HTML file. The tool is based on recursively applying the Jacobson-Vo [JV92] heaviest HCS (Heaviest Common Subsequence) algorithm to the different depth levels of the HTML parse trees. First, the children of the `<BODY>` tag of both files are compared. Using the HCS algorithm the children are aligned and corresponding tags in both files are identified. Then, taking the recursive step, the children of aligned tags are compared, just as the children of the `<BODY>` tag were compared in the first step.

Being based on HCS, which is a generalization of the LCS (Longest Common Subsequence) algorithm (see e.g. [Mye86]), TopBlend does not recognize move operations. Furthermore, in an apparent contradiction with what was said previously, TopBlend seems to reduce a structural comparison problem to a linear one. However, from the description of the algorithm in [CDH00] it can be seen that by moving subtrees around, thus changing the weight of a node significantly without major structural changes, you can create any matching at the first depth level. This regardless of the actual structural similarities between the blocks.

These arguments aside, TopBlend is in practice a very able tool for HTML differencing.

XMLTreeDiff

XMLTreeDiff by IBM Alphaworks [IAw01] is a general-purpose differencing tool for XML files. The tool is based on the work by [Tai79] and supports three basic node operations: updating, deleting and inserting a node. In addition, in order to increase the conciseness and readability of the diff, the tool has three aggregated operations: subtree prune (deleting an entire subtree), subtree graft (inserting a subtree) and subtree move (moving a subtree) [IAw99].

The output of the tool is encoded in either of two XML-based update languages, both using the standard XPath syntax for identifying nodes in the source document.

The largest flaw of the tool appears to be that the potentially useful move operation is not detected in all cases. For instance, when replacing a subtree rooted at a node a with a new node b so that the subtree rooted at a a becomes a child of b (e.g. $(R;\ (a;\ a_1))$ is transformed to $(R;\ (b;\ (a;\ a_1)))$), XMLTreeDiff treats this as grafting a new subtree rooted at b.

At the time of writing, neither the source code of the tool nor a description of the operation aggregation method is available.

LaDiff

LaDiff [CRGW96] is a structural differencing tool for LaTeX documents. In addition to handling updates, inserts and deletes, LaDiff also handles moves. The move operation is built into the LaDiff core algorithm, as opposed to the move operation of XMLTreeDiff. In practice this means that LaDiff identifies moves that XMLTreeDiff would treat as a pair of prune and graft operations.

The program is more of a demonstration of the algorithms developed in [CRGW96] than a full-blown diff tool: only certain LaTeX elements are supported. However, the main interest in this tool from the point of view of this thesis are the underlying algorithms. The differencing algorithm of LaDiff is discussed further in section 4.4.

4.3 Merge tools

UNIX diff3

The UNIX diff3 tool is a front end to the diff tool which provides support for 3-way differencing and merging of text files.

Assume we have a base document B, of which there are two branches B_1 and B_2. Diff3 accomplishes merging by either producing the diff $B \ominus B_1$ and patching B_2, or by patching B_1 with the diff $B \ominus B_2$.

Being based on diff, the tool is inherently a plain text merge tool, and as such is not suitable for structural merging (as pointed out in section 4.2, plain text and structural differencing are fundamentally different).

IBM Alphaworks' XML Diff and Merge Tool (XMLDiff)

IBM's Alphaworks have also developed a "XML Diff and Merge Tool" (XMLDiff) [IAw01b] independently of XMLTreeDiff. The tool compares two XML input files, and shows the differences. The user is then able to interactively merge the two versions by choosing either branch at the differing locations.

The implementational details of the tool are not publicly available, but it appears (according to [Bri99]) that the XML parse trees are compared one depth level at a time. For instance, when comparing the trees *(R; (a; b c))* and *(R; (d; (a; b c)))* (i.e. an additional node d has been added between the root and a) the tool treats this as if the subtree *(a; b c)* has been removed, and a new tree *(d; (a; b c))* inserted.

Node equality comparison is based on matching of node ID:s (if present) or the node tag name and content [Bri99].

Being a pairwise comparison tool XMLDiff cannot automatically perform merges. It's usefulness in the scope of 3DM is also limited due to the closed implementation.

Various application-specific merge tools

In addition to the above mentioned general merge tools there are numerous application-specific merge tools, either integrated in the application or standalone. Examples include the "Compare Document..." tools in Microsoft Word and StarOffice and programs for synchronizing appointments between a PDA and your laptop.

These tools are of little relevance to 3DM. The level of merging usually limited to line-by-line comparison, they are proprietary and application-specific and, as far as the author is aware, they do not exhibit any new thinking in merging algorithms.

4.4 Algorithmic research

The 3DM tool draws on algorithmic research in two areas:

- Tree differencing, matching and tree edit histories
- Tree merge algorithms

The research areas cover the three aspects of the system: tree matching, differencing and merging. In the following sections the current research in algorithms relevant to 3DM from these areas is reviewed.

4.4.1 Tree differencing algorithms

Suppose we have two trees T_1 and T_2, and that T_2 has been obtained by editing or transforming T_1 in some unknown way. The tree differencing problem is to recover the set of changes between T_1 and T_2 so that this set of changes can be used to obtain T_2 from T_1.

The tree differencing problem is typically approached by defining a set of operations to express the changes (e.g. node insert, delete, and update) and an optimality criteria on the set of changes. Intuitively, the optimality criteria allows us to express that we want to have a meaningful set of changes. For instance, assuming that $T_1 =$*(R; a b)* and $T_2 =$*(R; a (b; c))* a set of changes deleting all nodes in T_1 and then inserting all nodes in T_2 is not desirable, whereas a set expressing that c is inserted as a child of b is.

The tree differencing problem is a generalization of the string (or ordered sequence) comparison problem, in the sense that strings can be thought of as ordered trees of height 1 [Cha99, p. 14]. Although techniques from the domain of string comparison can be applied to the tree differencing problem string comparison algorithms are not directly suitable for tree differencing [ZS97, Cha99, p. 14]. The possibility that the tree differencing problem can be reformulated in the string domain is, however, still not formally ruled out [ZS97].

Tree differencing can be divided into two subcategories dealing with ordered or unordered trees. In unordered tree differencing, the order of child nodes is not important. For instance, if the trees *(R; a (b; c d))* and *(R; (b; d c) a)* are considered unordered, there is no difference between them. Even simple variants of the unordered tree differencing problem has been shown to be NP-hard [ZS97, Cha99, p. 17].

In 3DM, we share the assumption of [CDH00, IAw01, CRGW96] that the trees are ordered. The assumption is based on computational feasibility, the likely ordered nature of the input (e.g. text documents) and the fact that XML has no inherent support for expressing when element children should be considered ordered or unordered.

The first non-exponential algorithm for solving the ordered tree differencing problem is due to [Tai79], who also introduced the concept of edit distance to measure the difference between two trees [ZS97]. As the majority of subsequent work (e.g. [ZS89, ZS90, CRGW96]) on tree differencing is based on this distance, we shall examine it in some detail. At the same time we will define the related concepts of edit script and matchings between trees.

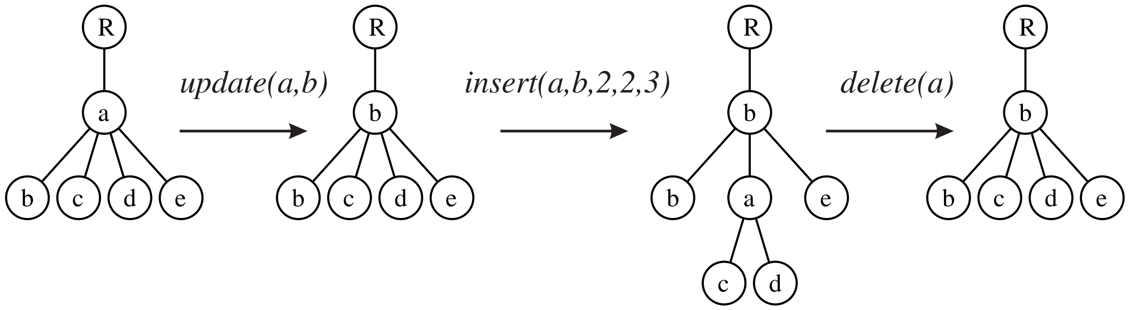


Figure 4.1: Illustration of the tree edit operations *update*, *insert* and *delete*.

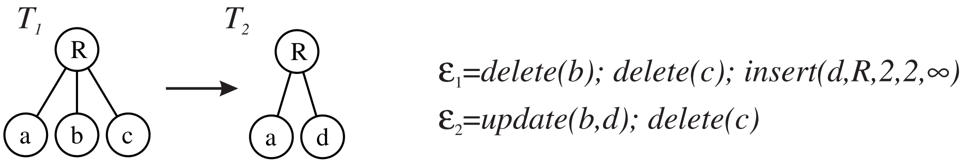


Figure 4.2: Both the edit scripts $\mathcal{E}_1$ and $\mathcal{E}_2$ transform T_1 into T_2. Assuming unit cost for all operations, the script $\mathcal{E}_2$ costs less, in this case 2 units. It is also the minimum cost edit script that transforms T_1 into T_2 and thus the edit distance is 2.

4.4.2 Edit distance and scripts

Suppose T_1 and T_2 are trees and that we have defined a set of tree edit operations. For instance, the edit operations may be the commonly used node insert, delete and update operations (see e.g. [Tai79, ZS89]):

$delete(x)$ Delete the node x, and insert the children of x between the left and right neighbors of x.

$insert(x, y, n, s, t)$ Insert the node x as the n:th child of node y, making the children s trough t of y children of x. This is the inverse of the $delete(\cdot)$ operation.

$update(x, y)$ Updates the content of x to y.

Examples of these operations are shown in figure 4.1.

Let us also assign a cost c to the edit operations. The cost of an edit may depend on its argument. e.g. updating a node whose contents is "Tux" to "Linux" would cost less than updating to "Windows".

An *edit script* $\mathcal{E}$ that transforms T_1 to T_2 is a sequence of edit operations, so that when starting with T_1 and applying each operation from the sequence is applied in turn, T_1 is transformed into T_2. The *edit scrip cost* is the sum of the costs of the individual operations in the script. [ZS97]

The *edit distance* between two trees is the minimum cost of all possible edit scripts transforming T_1 into T_2. Figure 4.2 illustrates the concept of edit distance.

4.4.3 Tree Matchings

Edit scripts based on the above mentioned operations give rise to the notion of a *matching*[2] between the trees. Intuitively, a matching shows the correspondence between the nodes in the trees (as shown in figure 4.3). Matched nodes (i.e. connected by a matching edge in figure 4.3)

[2]The term *mapping* is also used. [Cha99] uses matching, and [ZS97] uses mapping.

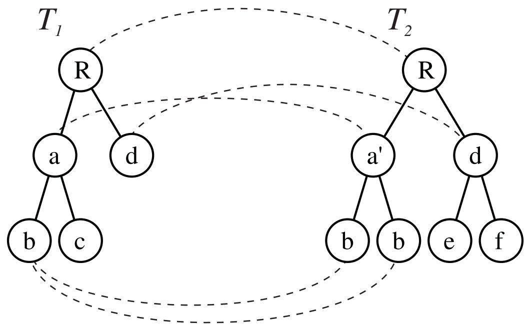


Figure 4.3: A matching gives the correspondence between the nodes in two trees. Note that a matching need not be one-to-one, as illustrated by node *b* in the left tree, which has two matches in the right tree.

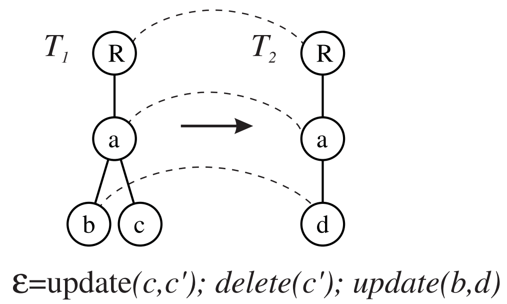


Figure 4.4: An edit script that cannot be recovered from the corresponding matching. The update in the beginning of the script cannot be recovered, as it leaves no trace in the matching, due to the deletion of the updated node in the second step of the script.

either represent no operation or and update operation, depending on whether the content of the nodes match. Unmatched nodes in T_1 correspond to delete operations, and unmatched nodes in T_2 to insert operations. Matchings can be restricted to be one-to-one, but may also allow more general (e.g. one-to-many) relationships.

Matchings are also useful when using a larger set of edit operations, such as one including subtree move and copy operations. In [CG97b] methods for transforming a matching to an edit script and vice versa using insert, delete, update, copy, move and "uncopy"[3] operations is sketched. The full details of the methods are given in [Cha99, pp. 90-118].

In general, a one-to-one mapping between matchings and edit scripts does not exist. In other words, matchings and edit scripts are not equivalent representations, unless some restrictions of well-formedness are imposed on the edit script and/or the matching. Figure 4.4 illustrates an edit script, which cannot be recovered from the corresponding matching, and figure 4.5 illustrates a matching that cannot be expressed using the three basic insert, delete and update operations.

The original tree differencing algorithm by [Tai79] was quite complicated and had high space and time complexities [ZS97]. In [ZS89] Zhang and Sasha presented an algorithm calculating the edit script corresponding to the minimum edit distance in time $O(|T_1| \cdot |T_2| \cdot depth(T_1) \cdot depth(T_2))$. A slightly faster variant, based on the assumption that all edit operations have unit cost was presented in [ZS90]. Both algorithms use inserts, deletes and updates as basic operations.

In [CRGW96] an algorithm based on insert, delete, update and move operations is presented. By making certain assumptions about the uniqueness of nodes and allowable parent-child relationships

[3] The operation is referred to as the "glue" operation in the paper

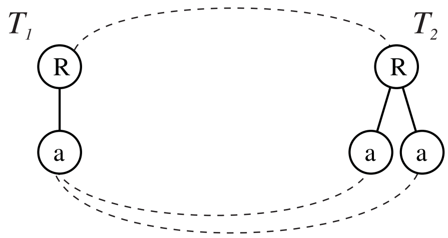


Figure 4.5: Matching that cannot be expressed using insert, delete and update operations. Using these operations we can only express the copy of node a as an insert, not a copy of the original a.

the algorithm achieves a time complexity of $O(max(|T_1|, |T_2|) \cdot e + e^2)$, where e is the weighted edit distance between T_1 and T_2. The algorithm consists of two phases: building a, as the author calls it, "good matching" between the nodes of T_1 and T_2, and finding the minimum-cost edit script corresponding to the matching.

A heuristic tree differencing algorithm using a rich set of operations (insert, update, delete, move, copy and uncopy) is presented in [CG97b]. By imposing certain, intuitively meaningful, restrictions on the edit scripts the minimum cost edit script problem is transformed into the matching domain. The starting point of the algorithm is a matching, where each node is matched to each node in the other tree. Edges are then pruned from this matching using a set of well-designed pruning rules. When no further pruning can be performed, the best matching is approximately solved by finding the minimum-cost edge cover for a bipartite graph consisting of the nodes in T_1 and T_2. The algorithm yields a solution in $O(n^2)$ time in most cases, with a worst-case performance of $O(n^3)$.

By making different assumptions regarding the available edit operations, the cost of the operations and the tree structure other tree matching algorithms have been designed (e.g. [Lu79, CG97a]). A common trait in all cases is that they are based on finding a matching or edit script corresponding to the minimum edit distance. Fast algorithms, with time complexities less than $O(n^2)$, have so far only been obtained by restricting the allowable matchings and edit operations, making assumptions about the tree structure, or by using heuristic solutions.

4.4.4 Tree merging algorithms

Suppose we have three trees T_B, T_1 and T_2. T_1 and T_2 have both been obtained from T_B trough editing or some other transformation: T_B is thus the base version of which T_1 and T_2 are branches. No record of the edits or transformations exist, only the final trees, and we want to construct the 3-way merge of T_B, T_1 and T_2. This is the tree merge problem, as defined in section 3.3.

Recall that an edit script transforming a tree T into another, T', may be obtained by running a structural differencing algorithm (see section 4.4.1) on the trees T and T'. We can then use the edit script as the edit history between T and T', and may thus also consider merge algorithms based on edit histories, although they are not explicitly available. Algorithms based on merging T_1 and T_2 without at least implicit knowledge of T_B or the edit histories are not considered here, because such algorithms cannot perform automatic merging.

In [Ask94] a method for merging unordered trees and identifying conflicts is outlined. In addition to the base and branch trees the method uses the edit scripts that transform the base version into the branches. The edit operations allowed in the script are node insert, delete and update. As the method is one of the few references to a general structured merge algorithm that the author is aware of, we will now examine it in some detail.

Let the edit script $T_B \ominus T_1$ be $\mathcal{E}_{B1}$ and the script $T_B \ominus T_2$ be $\mathcal{E}_{B2}$. In the first step, we add information to each node in T_1 and T_2, indicating if the node has been inserted, deleted or changed compared to the base version. The paper does not give any details on how this is implemented, but it could be accomplished by for instance running the edit scripts $\mathcal{E}_{B1}$ and $\mathcal{E}_{B2}$ on T_B, and keeping track of the modified nodes. If a node has modified children, it is itself also considered to be modified. This way modifications are reflected all the way to the root of T_1 and T_2.

We then continue by comparing the roots of T_1 and T_2. If the content of either root has changed, the changed content is used. If both roots have changed, a conflict has occurred.

Next, we compare the children of the roots. Any new (i.e. inserted) children of the root in either tree are also inserted in the merged tree, and conversely any child of either root that is deleted is removed from the merged tree. For instance, if the root of T_B has the children $a\ b\ c$ and the roots of T_1 and T_2 have the children $a\ b\ c\ d$ and $b\ c$ respectively, the merged child list is $b\ c\ d$.

The method can now be applied recursively to the children of the root in the merged tree. The paper does not state specific details of the recursive step, but the general idea should be clear.

The method described in the paper handles moves and copying of nodes as node insert and delete operations, and thus lacks the ability to successfully handle the move/update case.

The paper furthermore presents a method for detecting conflicts in the context of the merge method described above, and also proposes how merge conflicts should be handled. For instance, if a node is deleted in one tree, and changed in the other, how should this be handled? The author suggests that one of the branches could be used as a main alternative, whose edits are always carried out in the case of a conflict.

A similar method of merging structural data is sketched in [MD94], having the same shortcomings regarding node move and copy operations.

In [BSV98] a system for merging and conflict resolution, called MARC, is presented. MARC operates in the context of a CSCW system named CoNus [Berger96], whose data model consists of objects with attributes and links between these objects. The links can be of two types: hierarchical or semantic. The hierarchy links connect the objects into a tree structure.

MARC is based on edit histories. When two object trees need to be synchronized, the edit histories gathered since the branch point are used. The edit histories are then analyzed in order to detect and resolve conflicts. Based on the analysis and conflict resolution, two new edit histories are subsequently produced, which when applied to the trees bring them into sync (i.e. in effect a merge).

Compared to the merge methods presented in [Ask94] and [MD94], the one used in MARC treats the tree vertices (i.e. the hierarchical links between the objects) as entities that may be created and deleted. This allows for a more powerful merge that can handle the move/update case. On the downside, copies are not handled and the merged structures may be forests rather than trees.

A system for comparing and querying structural documents is briefly presented in [WSC97]. The system is based on a insert-delete-update tree differencing algorithm, but also supports moves trough post processing of the algorithm output. The paper mentions that the system has two merge-related operations: $mergeable(T_B, T_1, T_2)$ for determining if a 3-way merge is possible, and $merge(T_B, T_1, T_2)$ to perform a 3-way merge. Unfortunately no further details on this system seem to be available.

Further research on structural merge includes [HPR89], in which a semantic merge of computer programs is described. The initial setup is similar to the structural merge problem: We have three versions of a program P_B, P_1 and P_2, where P_B is the base program and P_1 and P_2 are branches of that program. The paper defines what changes in a program are and the semantic merge[4] of the programs. Roughly, the change definition states that if the value of an output variable x is different between two programs P and P' for some input value of the programs, the computation of x has changed. The (noninterfering) semantic merge of P_B, P_1 and P_2 is then a program that contains all changes $P_B \ominus P_1$ and $P_B \ominus P_2$.

A "flexible merging framework" is the topic of [MD94]. Flexible means that it should be possible to use different merging policies depending on the situation. Some high-level requirements for a flexible merging tool are also presented. In short, the requirements are: 1) the ability to produce automatic diffs, 2) to allow user interaction with the merging process, 3) to operate on general objects, 4) to allow semantics based merge and conflict detection and 5) to let the user specify a fine-grained merging policy (including automatic conflict resolution). Finally, the authors show how to emulate the behavior of several merging tools using their representation of merging policies.

It seems like there has not yet been any attempt at defining the general structural merge algorithm that meets the requirements on the 3DM tool (with the possible exception of [WSC97]). One probable reason is that it has been generally thought that an understanding of the semantics is needed to perform a successful merge. Another reason is that the merging tends to be specific to the particular system it is designed for, as can be seen from the presented applications and algorithms. Factors affecting the design of the merge tool include 1) is edit history available, 2) what are the edit operations (if they are available) and 3) how is the data represented?

In this thesis I propose that there is in fact little need for a semantic merge, assuming that the tree encoding exposed to 3DM reflects the internal data structures of the application (see section 3.3).

4.5 Discussion of related work

The existing merging and differencing tools are evaluated with respect to the requirements set on 3DM in sections 3.3 and 3.4. The evaluation with respect to differencing requirements is presented in table 4.1 and the evaluation with respect to merging requirements in table 4.2.

When looking at table 4.1 we can see that there currently exists no tool or algorithm that satisfies all requirements. Support is lacking in the areas of patching and support for copy and move operations (support for these operations save space when large subtrees have been moved or copied). The algorithm presented in [CG97b], although it lacks in efficiency, seems to be the best candidate, and could be extended with the appropriate encoder for the diff (in this case an edit script) as well as a complementing patching algorithm. However, as we will see in chapter 8, the architecture of 3DM allows us to add efficient differencing and patching capabilities with little extra effort compared to merging only, and hence the approach in [CG97b] was discarded in favor of the novel tree differencing approach of chapter 8.

In table 4.2 the reviewed merging tools and algorithms are evaluated according to the merging requirements from section 3.4. The only candidates supporting structural 3-way merging are the algorithm in [Ask94] and the one used in MARC. The only candidate that comes close to satisfying

[4]the term used in [HPR89] is *program integration*.

Capability	CVS/diff	Notes/Groupwise	Groove	SyncML	WebDAV	TopBlend	XMLTreeDiff	LaDiff	XMLDiff	[ZS89]	[CG97b]
Tree differencing	-	0	0	-	-	0	+	+	+	+	+
Diff supports copy and move	-	?	?	-	-	-	-	-	-	-	+
Tree patching	-	0	0	-	-	-	-	-	-	-	-
Encoding of tree diffs	-	0	0	+	-	0	+	+	-	-	-
No additional input except XML files	+	-	-	-	+	+	+	+	+	+	+
Works (or can be made to work) with XML	+	0	+	+	+	+	+	+	+	+	+
Nonproprietary	+	-	-	+	+	+	-	+	-	+	+
Resonably efficient	+	+	+	+	+	+	-	+	?	+	-

Table 4.1: Capabilities of tools and algorithms with respect to the differencing requirements. The presence of a capability is indicated by a +, the absence by a –. 0 indicates that the capability exists to some extent, but not in a way useful for 3DM. ? indicates that the status could not be determined.

Capability	CVS/diff3	Notes/Groupwise	Groove	SyncML	WebDAV	MS Word/StarOffice	CMLDiff	[Ask94]	MARC [BSV98]
Structural merge	-	0	+	-	-	-	0	+	+†
Merge tracing	-	?	-	-	-	+	+	+	+
3-way merge	+	-	-	-	-	-	-	+	+
Subtree move	-	-	0	-	-	-	-	-	+
Subtree copy	-	?	?	-	-	-	-	-	-
Handles move/update	-	-	0	-	-	-	-	-	+
No additional input except files	+	-	-	+	+	+	+	-	-
Works (or can be made to work) with XML	+	0	+	+	+	-	+	+	+
Nonproprietary	+	-	-	+	+	-	-	+	+
Decoupled operation	+	+	+	+	+	+	+	+	+
Peer-to-peer synchronization	+	-	-	-	0	+	+	+	+
Optimistic locking	+	0	+	+	+	+	+	+	+
Efficient	+	+	+	+	+	+	+	+	+

† MARC handles general graphs

Table 4.2: Capabilities of merging tools and algorithms with respect to the merging requirements. The symbols have the same meaning as in table 4.1.

the essential requirements is MARC [BSV98], but it requires an edit history (which we would thus have to generate with a tree differencing algorithm) and lacks support for the copy operation.

Hence, MARC will not be used here, mainly do the need for an edit history. Although algorithms exist for generating edit scripts from matchings ([CG97b, Cha99, pp. 97–119]), and they could be extended to handle the the three very useful types of matches (content, structural and full) we introduce in section 5.2, we would still have to extend the algorithms of MARC to handle node ordering (with possible conflicts), correct dispatching of edit operations to each copy as well as checks for well-formedness to verify that the result is indeed a tree.

We can thus conclude that the merging algorithms and tools that the author is aware of cannot be utilized, and that there is a need for a novel merging algorithm. We will start developing the algorithm in the next chapter.

Chapter 5

Defining Tree Matching and Merging

We approach the tree merge problem by dividing it into two independent subproblems: 1) matching of trees and 2) merging of matched trees. The base version of the tree will be denoted by T_B, the branches with T_1 and T_2, and the merged tree with T_M.

The problem of matching trees is concerned with constructing a matching between two trees T and T', indicating the node correspondence between the two trees (see 4.4.1). The matching problem may be trivial, as in the case when the corresponding nodes in T and T' are tagged with the same node ID, or considerably harder, as in the case where nodes in T and T' are not tagged nor unique. See figure 5.1.

The tree matching may be either implicit or explicit. An explicit matching is typically expressed as edges between the nodes of the trees T and T', whereas an implicit matching may be an edit script (recall from section 4.4.1 that it is possible to construct a matching from an edit script), or an assumption that only nodes on the same depth level in the trees match (i.e. the for any nodes $n \in T$ and $m \in T'$ that match, the children of n may only match children of m and vice versa) as in the TopBlend tool.

Without a matching (either explicit or implicit) between the trees it is impossible to perform an unambiguous 3-way merge. For instance, suppose T_B=*(R; a b)* T_1 =*(R; a' b)* and T_2 =*(R; a)*. How can we tell that $T_B(a)$ was updated to $T_1(a')$ and that $T_B(b)$ was deleted, if we don't know that $T_B(a)$ matches $T_1(a')$ and $T_2(a)$, and that $T_B(b)$ matches $T_1(b)$ but corresponds to no node in

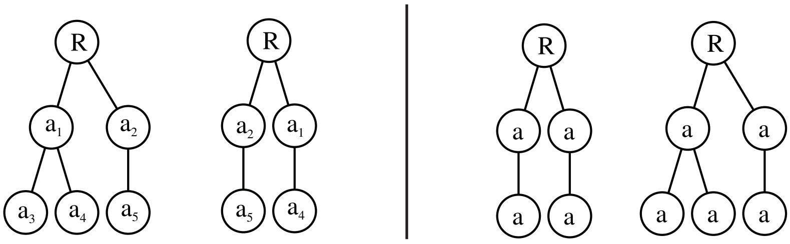


Figure 5.1: Easy and hard matching problems. Matching the trees on the left is easy, as unique IDs (indicated by subscripts for the node labels) are available. Matching the trees on the right is considerably harder.

T_2? Or suppose that it is actually $T_B(b)$ that matches $T_2(a)$. In that case $T_B(a)$ has been deleted in T_2 and updated in T_1 — a possible conflict situation.

The problem of merging matched trees deals with the structural tree merge problem with the additional assumption that a matching between T_B and T_1 as well as T_B and T_2 exist.

The subproblems are independent in the sense that, given some common constraints on the matching, any matching and merging program modules adhering to the constraints may be combined freely.

We will now proceed by defining the desired behavior of the 3-way merging tool, and then analyze this behavior to extract requirements for the matching and merging algorithms in 3DM. In the process, we will state several useful definitions related to tree matching and merging.

5.1 Desired behavior of the 3-way merging tool

The definition of desired merging behavior is based on the use cases presented in chapter 2, as well as a relatively large number of small hand-written "merge cases". We present the merge cases in the next section, and then proceed to the merging definition.

5.1.1 Merge cases

The merge cases were designed to be simple illustrations of desired merging behavior, and to give a better coverage of merging situations than is possible with only a few use cases. Each merge case consists of four small XML files: the base version, two branches, and the desired 3-way merge of the base and the branches.

Furthermore, as the merge cases are written in XML, they can be used to test the final implementation of 3DM, along with the use cases.

All in all 42 merge cases were written. In table 5.1 we present a selection of the cases that illustrate the desired merging behavior in common as well as some of the more tricky situations. For the XML source of each case, see appendix G.

5.1.2 Functional requirements for 3-way merging

We start our definition of tree merging by gathering the requirements from the use cases, the merging cases and the problem statement and the architectural requirements (sections 3.3 and 3.4). These top-level requirements will be used in sections 5.2 and 5.3, when stating the specific requirements on the matching algorithm and the 3-way merging algorithm.

1. The operation of tree merge should be easy to understand. We will rather have an algorithm that works in 90% of the cases, than one that works in 95%, but leaves the user uncertain about the correctness of the result.

2. Any change (with respect to the base tree) to a node in either branch, be it move, delete, update, insert or copy, should be present in the merged tree. Move and copy operations should be unrestricted (a restriction would be that e.g. a node may not be moved under a different parent).

3. Node operations should be considered relative rather than global. For instance, if the children of a node n are reordered in one branch, and the subtree rooted at n is moved in the other

Case	Description
D1	The case illustrates how deletes at different locations in the branches are merged together.
I2	In this case, a new item is appended to the child list of the root in both branches. The proposed way of handling this situation is to append both items to the child list in the merged tree. A conflict warning should be generated, though.
U2	Here, the node b is updated to b_1 in both branches. Although this is in some sense a conflict (two authors have edited the same piece of information), the merge is easy to construct, as the updates are identical.
M2	An example of moves made to the same child list from both branches. This could for instance be a text, where the paragraphs have been reordered.
M3	The case illustrates moves made in both branches. In the first branch, the entire subtree T_a is moved and in the second, the children of a have been moved. The merge tool should be able to handle this type of case, where we move an entire subtree in one branch and modify the structure of the subtree in the other.
M4	The node a is moved to different loctions in the branches. The tool should detect such conflicts.
C4	Here, nodes (b and c) are copied to the same destination in the branches. A possible solution, as illustrated by the case, is to insert both nodes after each other at the destination. As in case I2 we want to be warned about this situation.
A3	In this case, subtrees are reordered (T_{s_2} and T_{s_3} are swapped) as well as copied (T_{s_1}).
A4	This is an example of a subtree being copied (the subtree T_a), but where the structural changes in the original subtree should not be propagated to the subtree of the copy.
A11	Shows and combination of inserts in the first branch and moves in the other, both targeting the same child list.
A14	On the contrary to case A4, in this case changes are propagated to the copied nodes, i.e. both copies of b are updated.
X1	As in case U2 the same node is updated in both branches. Unlike in U2 the updates are not equal, so a conflict should occur.
X4	The case is an example of conflicting moves.
X5	In this case, a subtree that has been modified is deleted (the node e in the subtree T_c was updated). We want the merge tool to warn about such “lost edits”.

Table 5.1: Selected merge cases that illustrate desired merging behavior.

branch, both the move of n and the child reordering should be present in the merged tree. Thus, the children of n is moved *relative to* n, not to absolute positions. See merge case M3.

4. By moving, copying or inserting a node, we put it into a context, consisting of the nodes surrounding it. We want the context to be preserved in the merged tree.

   Consider a text document where a paragraph is moved. It makes sense to retain (at least) the paragraphs immediately before and after the moved paragraph in the merged document. Allowing the paragraph before or after to be moved in the other branch would most likely result in an meaningless text.

   Putting it differently, this requirement states that structural modifications should not be allowed too close to each other if they are not made on the same branch. Under the assumption that nodes close to each other may have strong semantic dependencies, such as paragraphs in a document, this requirement makes sense, and it is not desirable to perform an automatic merge. See merge cases M2, A11 and X4. Merge case A3 is an example of a merge situation that cannot be handled if we require the context to be preserved.

5. If a subtree is copied in one branch, and modified in another, the modifications should propagate to all copies of the subtree in the merged version. An example of this is two image

files, the first containing a texture and the other a drawing using the texture. If the texture changes, the changes should be reflected to all places in the drawing where the texture is used.

There are, however, counterexamples to this. If the copied subtree is small, or the copies only match their original approximately, modifications should not necessarily be propagated. An example of this is a text document with nodes for each word. When inserting new text, it's hardly desirable that the inserted words are considered new copies of previous instances of the word, because a change of a word (such as "the") in one paragraph could then be reflected to all other paragraphs.

For examples of node copying see merge cases A14 and A4.

6. Assume a node n exists in both branches, and that, in both branches, new nodes are appended as children to it (e.g. T_B=*(R; a)* T_1 =*(R; a b)* and T_2 =*(R; a c)*). In some cases, we want the merged child list to contain the nodes appended from both branches (e.g. the appended nodes are items in a to-do list; T_M =*(R; a b c)* or T_M =*(R; a c b)*), in other cases this is a conflict situation (e.g. a new paragraph is appended to the end of a section in both branches). For examples, see merge cases I2 and C4.

7. The 3-way merge should be symmetric, i.e. the merge result should not depend on if we merge the changes $T_B \ominus T_1$ into T_2, or the changes $T_B \ominus T_2$ into T_1. This requirement is also motivated by the easy-to-understand criteria: asymmetries in the merge are unintuitive. Some deviations from the symmetry requirement can however be allowed, provided they are understandable and the user is warned about them. For instance, if appends are allowed as suggested in the requirement 6, the merge will not be totally symmetric.

8. The following conflicts should be detected (at least).

   (a) Update/Update. The conflict occurs if the same node is updated in both branches. E.g. the same paragraph has been updated in both branches. Merge case X1.

   (b) Delete/Move, Delete/Update and Delete/Copy. This situation occurs if a node is deleted in one branch and "modified", i.e. moved, copied or updated, in the other. This is not necessarily a conflict, i.e. the node can still be deleted, but the user should be informed of the situation. Merge case X5.

   (c) Move/Move. The node is moved to different locations in the branches. Merge case M4.

5.2 Tree matching definitions and requirements

Consider the trees in figure 5.2. A matching between the nodes of the trees (see section 4.4.1) is a set of edges that connects the corresponding nodes in both trees, where correspondence can measured by e.g. comparing the contents of the nodes ([ZS97, CRGW96]). The matching between the trees T and T' in figure 5.2 is depicted in figure 5.3.

Matching A matching between two trees T and T' is a set of edges (n, m), $n \in T$ and $m \in T'$. Matchings are denoted with the letter $\mathcal{M}$, with optional subscripts identifying the trees involved (e.g. $\mathcal{M}_{TT'}$).

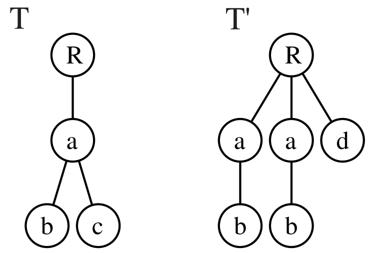


Figure 5.2: Two unmatched trees

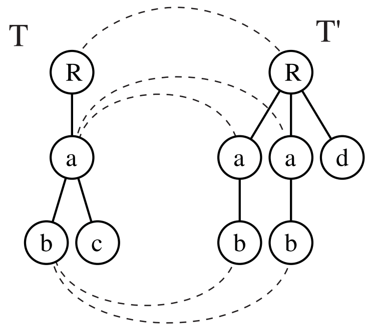


Figure 5.3: Matching between the trees T and T' from figure 5.2.

Matched nodes Given a matching $\mathcal{M}_{TT'}$, the nodes $n \in T$ and $m \in T'$ are matched iff the edge $(n, m) \in \mathcal{M}_{TT'}$.

For the purpose of merging we are interested in using the matchings to detect changes that transform one tree into another. By inspecting the matching in figure 5.3 we can obtain the changes that transforms T into T': The subtree rooted at $T(a)$ occurs twice in T' and has thus been copied; the node $T(c)$ in the base tree has no match and is thus deleted; in the target tree $T'(d)$ is also unmatched, which means it has been inserted. Once we have obtained the changes between trees, we can perform a merge by integrating the changes in a single tree.

When we are merging two branches, we check each node in the base tree and the branches for changes. Each change to a node is then propagated to the merged tree. Unfortunately, there are some cases where it is not possible to unambiguously determine the changes to a node. Consider the matching between the trees T_B (the base tree) and T_1 (the branch) depicted in figure 5.4. What is the change made to $T_B(a)$? $T_B(a)$ is matched to three nodes, each giving a different message: the first indicates no change, the second that the children (b and c) of a have been replaced with two new children (d and e), and the third that the content of a has been modified to a'.

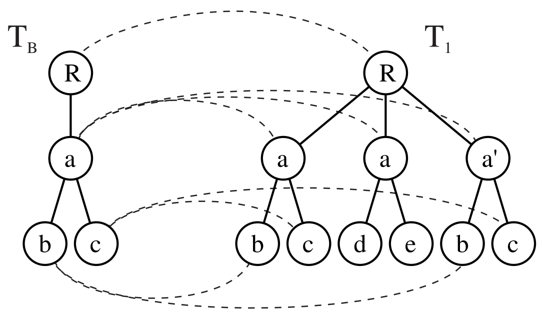


Figure 5.4: Matching between trees T_B and T_1

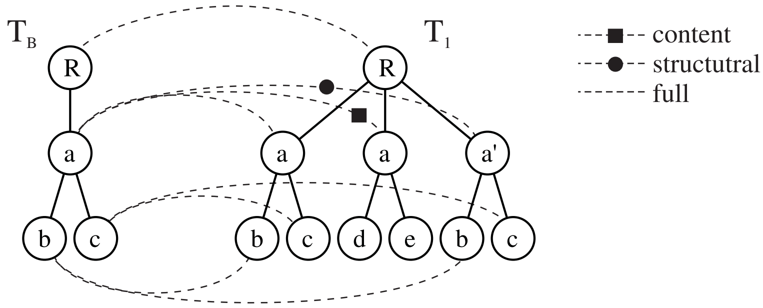


Figure 5.5: Typed matching between the trees in figure 5.4.

The solution used here is to annotate the matching edge with a *match type*. The match type determines what type of change we should look for when comparing the matched nodes: changes in the content of the node, changes in the child list of the node, or both. When changes in the content of a node n should be propagated to instances of n in the merged tree originating from the other branch, we say that the node is content matched, and when changes to the child list of n should be propagated we say that the node is structurally matched. If two nodes are both content and structurally matched, they are said to be fully matched:

Content matched nodes Two nodes $n \in T$ and $m \in T'$ are said to be content matched if the edge $(n, m) \in \mathcal{M}_{TT'}$ is of type content match. The predicate $contentMatch(n, m)$ denotes content matched nodes.

Structurally matched nodes Two nodes $n \in T$ and $m \in T'$ are said to be structurally matched if the edge $(n, m) \in \mathcal{M}_{TT'}$ is of type structural match.The predicate $strcuturalMatch(n, m)$ denotes structurally matched nodes.

Fully matched nodes Two nodes $n \in T$ and $m \in T'$ are fully matched iff they are content matched *and* structurally matched.

The notion of different matching types allows us to handle node copies in both a propagate-changes and no change propagation-manner, implementing merging requirement 5.

Figure 5.5 illustrates the matching case from figure 5.4, but this time with matching types. In this case, none of the changes made to $T_B(a)$ should be propagated to instances of a in T_M originating from the other branch, since the second copy of a in T_1 is content matched (i.e. the changes to the child list should not be propagated) and the third copy is structurally matched (i.e. the change of content to a' should not be propagated). This interpretation is natural if we think of the leftmost copy of a as the original instance of $T_B(a)$ in T_1 and the others as copies with some additional changes. Since the original instance has not changed, no changes should be propagated.

In order to simplify the merging algorithm we will impose the restrictions that each node in the target trees T_1 and T_2 have maximally one match in T_B. In practice, this means that we disallow the (somewhat counterintuitive) operation of "uncopying" (also known as gluing, see [CG97b]) several nodes into a single node. Figure 5.6 illustrates a matching where the node $T_B(a)$ is uncopied. A more natural way of seeing the relation between T_B and T_1 is that the a node to the right in T_B was simply deleted.

Another restriction we want to enforce on the matching is that every matched node in T_B must have a fully matched node in T_1. This corresponds to the notion that if a node from T_B exists

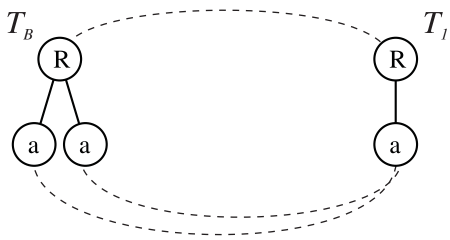


Figure 5.6: A matching with "uncopy"

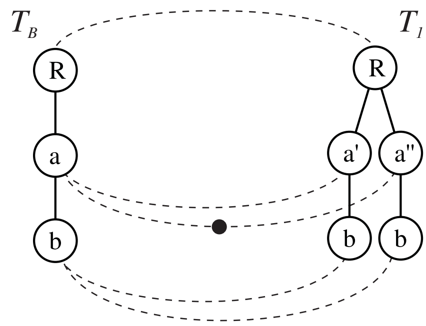


Figure 5.7: Matched trees T_B and T_1. We require that one of a's matches in T_1 is the original instance, in this case $T_1(a')$.

in T_1, one of the instances in T_1 should be considered to be the original from T_B, and the others are (possibly modified) copies. For instance, consider the trees depicted in figure 5.7 and assume both matching edges between the a nodes are structural matches. If such a matching were used one is tempted to ask what happened to $T_B(a)$? Two imperfect copies have been made, but was the original deleted? It seems more natural to think that the right child of $T_1(R)$ is in fact $T_B(a)$ which was modified, and that the right child is an approximate copy of $T_B(a)$. Hence the nodes $T_B(a)$ and $T_1(a')$ should be fully matched.

In relation to this restriction, we require that the changes to content and structure to propagate must be unambiguous. This means all content matches of a base node must reflect the same change in content, and all structural matches must reflect the same change in child list content.

When merging trees, we will use matchings with typed edges and that satisfy these three restrictions. We call such matchings natural matchings, the word natural emphasizing an natural and intuitive way of understanding the changes between the trees T_B and T_1.

Natural matching A matching $\mathcal{M}$ between two trees T_B and T_1 is said to be natural iff 1) each node $m \in T_1$ has 0 or 1 matches in T_B, 2) iff $n \in T_B$ has a matches in T_1, at least one of them is a full match, 3) iff $n \in T_B$ has matches in T_1, all the structural matches have identical child lists, 4) iff $n \in T_B$ has matches in T_1, all the content matches have identical content and 5) the edges $(n, m) \in \mathcal{M}$ are typed. The types are content match, structural match and full match.

What are the characteristics of a "good" matching between two trees? It is not sufficient to only consider node content equivalence, as illustrated by almost any XHTML document, which typically contains several `<p>` tags. If the tags are matched only according to content, matchings like the one depicted in figure 5.8 is possible. Such matchings cause the merging algorithm to treat the

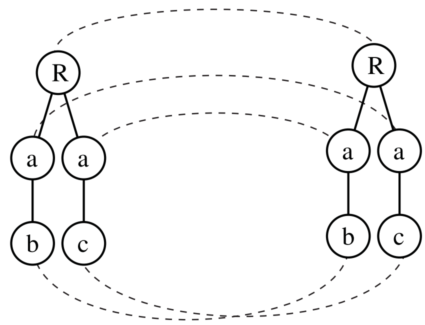


Figure 5.8: Matching constructed without considering the context of the a nodes, and hence they have been incorrectly matched.

paragraphs as if they have been moved around, when in fact they have not. Clearly, we also need to look at the context, i.e. neighbors, children etc. of nodes when building the matching.

Informally, the ideal matching is a matching that matches the "same" nodes in both trees. If the edit history transforming T_B to T_1 is available, such a matching can be constructed. However, in 3DM we do not assume the existence of any edit histories, and are therefore required to form the matching by looking at the trees T_B and T_1 only. As the trees do not contain any trace of the actual edit operations, the ideal matching is in fact sometimes impossible to deduce from the trees.

Ideal matching Assume two trees T_B and T_1 and that we know the edit history $\mathcal{E}_{B1}$ of insert, delete, copy and move operations that transformed T_B into T_1. The ideal matching of nodes between T_B and T_1 is obtained by tracking the nodes in T_B using the edit history. For instance, an inserted node is unmatched, a copied node is matched to the node that was copied etc.

Common approaches to tree matching include finding the matching that corresponds to a minimum cost edit script (using some predefined set of operations) or assigning costs to the matching edges, and searching for the matching which has a minimum cost [CG97b, CRGW96, Cha99].

Building a good matching between the trees is essential for the fulfillment of merging requirements 2, 3, 5 and 8. For instance, if a node is moved in one branch and changed in the other we want the merged tree to contain the changed node, moved to its new location. If the changed node and the original remain unmatched the merging algorithm will not operate correctly. Instead it will conclude that the moved node was deleted, which is a possible conflict, and furthermore, that a new node has been inserted.

In order for a merging to be successful it is fortunately not necessary that the matching is ideal. Had the node in the previous example not been moved in the second branch, the correct merging result would have been obtained although the nodes were unmatched, in this case because the result of the change operation and the delete, insert operations would have been identical. We say that a matching is well-formed, if the result of a merge is the same as if the matching were ideal.

Well-formed matching Let T_B be the base tree and T_1 and T_2 the branches. Let $\mathcal{M}_{B1}$ and $\mathcal{M}_{B2}$ be the ideal matchings between T_B and T_1 and T_B and T_2. The matching $\mathcal{M}$ is a well formed variant of $\mathcal{M}_{B1}$ ($\mathcal{M}_{B2}$) if the result of the tree-way merge of T_B, T_1 and T_2 using the matchings $\mathcal{M}$ and $\mathcal{M}_{B2}$ ($\mathcal{M}$ and $\mathcal{M}_{B1}$) is the same as the tree-way merge using the matchings $\mathcal{M}_{B1}$ and $\mathcal{M}_{B2}$.

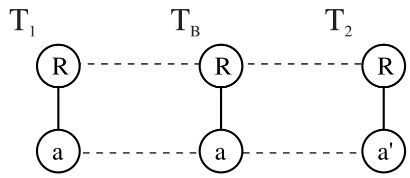


Figure 5.9: Partner nodes. The partners are $T_1(R)$ and $T_2(R)$ as well as $T_1(a)$ and $T_2(a')$.

Essential for the formation of a well-formed matching is that the nodes that have been edited in both branches are correctly matched, and that nodes edited in one branch only are not considered to have undergone changes in both branches according to the matching.

5.3 Tree merging definitions and requirements

We will now state definitions of concepts used when describing the 3-way merging algorithm, as well as give a more formalized version of the requirements in section 5.1, suitable as a specification for a 3-way merging algorithm. In all definitions, T_1 and T_2 are interchangeable, i.e. if a definition is stated for T_1 it holds for T_2, and if it is stated for T_1 and T_2, T_1 and T_2 can be interchanged in the definition. We also assume that the matchings $\mathcal{M}_{B1}$ and $\mathcal{M}_{B2}$ exist, and that they are natural (see section 5.2).

We start by defining the concept of partner nodes. Informally, we say that the "same" node in both branches are partners. Partner nodes are the alternatives for the node in the merged version. Figure 5.9 illustrates the concept of partner nodes.

Partners Two nodes $n \in T_1$ and $m \in T_2$ are partners iff there exists a node $b \in T_B$ such that n and b are matched nodes, and b and m are matched nodes.

Two more specific forms of partnership is content partners and structural partners. If two nodes are content partners, they represent the alternatives for the content of the node in the merged tree. If the nodes are structural partners, they represent the alternatives for child lists of the node in the merged tree.

Content partners Two nodes $n \in T_1$ and $m \in T_2$ are content partners iff there exists a node $b \in T_B$ such that n and b are content matched nodes, and b and m are content matched nodes.

Structural partners Two nodes $n \in T_1$ and $m \in T_2$ are structural partners iff there exists a node $b \in T_B$ such that n and b are structurally matched nodes, and b and m are structurally matched nodes.

Note that a matching (n, b), (m, b) where the first type is not structural and the second not content (or vice versa), i.e. the "mixed case", means that the nodes are not partners in any sense.

We continue by formalizing the notion of updated, inserted, deleted, moved and copied nodes:

Inserted node Assume $m \in T_1$. The node m is inserted (with respect to T_B) iff it has no match in T_B.

Deleted node Assume $b \in T_B$. The node b is deleted (with respect to T_1) iff it has no match in T_1.

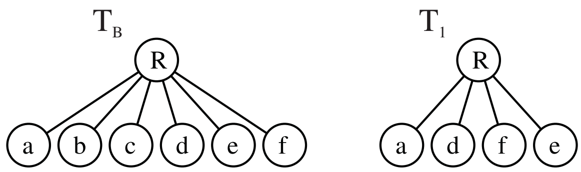


Figure 5.10: Illustration of *inSequence*. For T_1, the nodes a and d are in sequence with respect to the child list of $T_B(R)$, whereas f and e are not.

Updated node Assume $b \in T_B$ and $m \in T_1$. If m and b are content matched but their contents is not the same, or m and b are structurally matched, m is updated with respect to T_B.

Before we state the definitions of moved and copied nodes we introduce the following useful definitions:

Sequence number Assume n is a node in any tree. Each node n_i in the child list $\mathbf{n} = n_1, \ldots, n_k$ has a sequence number, denoted by $S_n(x)$. The sequence numbers are defined so that $S_n(n_i) = i$ and $S_n(x) = -1$, $x \notin \mathbf{n}$.

List partner Assume $b \in T_B$ and $m \in T_1$.Iff $\mathbf{m}(i)$ has one or several matches $y_i \in \mathbf{b}$, its list partner is the match y_i with the smallest sequence number. Otherwise $\mathbf{m}(i)$ has no list partner.

In sequence Assume $b \in T_B$ and $m \in T_1$. $\mathbf{m}(k)$, $k > 1$ is said to be in sequence with respect to $\mathbf{b}$ iff $\mathbf{m}(k-1)$ has a list partner x in $\mathbf{b}$ and $\mathbf{m}(k)$ has a list partner y in $\mathbf{b}$ and $S(x) < S(y)$ and for all i, $S_n(x) < i < S_n(y)$, $\mathbf{b}(i)$ is deleted in T_1. $\mathbf{m}(0)$ is in sequence with respect to $\mathbf{b}$ if it has a list partner x in $\mathbf{b}$ and for all i, $i < S(x)$, $\mathbf{b}(i)$ is deleted in T_1.

The definition of "in sequence" states that two consecutive nodes in a child list on either branch are in sequence with respect to the base tree, if they are also consecutive in the base tree, or all the nodes in between them have been deleted. Figure 5.10 illustrates this concept. We can now define what a moved and copied node is:

Moved node Assume $m_i \in T_1$, m_i is not inserted, the parent of m_i is m, and the match of m is $b \in T_B$. m_i is moved if m_i is not in sequence with $\mathbf{b}$. The node is *far moved* if m_i has no list partner in $\mathbf{b}$.

Less formally, a node is moved if it is has another parent (in which case we say it is far moved) or if it has another predecessor than in T_B.

Copied node Assume $m \in T_1$ and that the match of m in T_B is b. The node m is copied with respect to T_B if it is moved (with respect to T_B) and b has more than 1 match in T_1. If m is copied and several matches are present in a child list $\mathbf{m}_p$, then the match of m in $\mathbf{m}_p$ with the smallest sequence number is the *primary copy* in $\mathbf{m}_p$ and any other copies in $\mathbf{m}_p$ are *secondary copies* in $\mathbf{m}_p$. If a node $b \in T_B$ has matches which are copied nodes, we say that b is copied.

Note that the definition of a moved or copied node only depends on its parent node and position in the child list. Thus, if you move or copy an entire subtree only the root will be marked as moved/copied. This corresponds to requirement 3.

We will also need the concept of merged node, which are the nodes that make up the merged tree.

Merged node The nodes in T_M are merged nodes. Each merged node is the result of merging a single node (from either T_1 or T_2) or two nodes (from T_1 and T_2), in which case the nodes are partners. A merged node p that is the result of merging the node n (and m) is said to be the merged node of n (and m).

We also need to formalize what it means for a node to "stay in context":

Left (right) node context Assume $n_i \in \mathbf{n}$ and T_B is the base tree. The left (right) node context of n_i is $n_k n_i$ ($n_i n_l$), where n_k (n_l) is the first node preceding (succeeding) n_i in $\mathbf{n}$ that is not deleted with respect to T_B. If n_k (n_l) does not exist, the node has no left (right) context.

Node context The node context of n_i is $n_k\, n_i\, n_l$, where $n_k\, n_i$ is the left and $n_i\, n_l$ the right context. If either the left or the right context does not exist, the node context is only the right or left context. In these cases we say that the context is either left or right.

By using the formalized requirements presented below, we are able to state the content of the merged nodes, as well as the structure of the merged tree T_M.

First, we formalize requirements 2 and 4, that all updated, inserted, deleted, moved and copied nodes should be in the merged tree, and that their context should be preserved.

1. Assume $n \in T_1$ and $m \in T_2$ are content partners. If either node is updated, the content of the merged node should be that of the updated node. If both nodes are updated, a the content of the merged node should be the content merge (for a precise definition see section 7.1.3) of n and m. If neither is updated, the content of the merged node is the content of either node (there is no difference, since their contents are equal in this case).

2. Assume n is a deleted node. n should not appear in T_M.

3. Assume $n_i \in T_1$ is an inserted node and n is the parent of n_i. The node context of n_i in the child list of the merged node of n should be the same as the context of n_i in $\mathbf{n}$. If the context of n_i is a left (right) context it should appear as first (last) in the merged child list. Informally, an inserted node should always have the same (non-deleted) predecessor, successor and parent as it has in the branch it was inserted in.

4. Assume $n_i \in T_1$ is a moved node and n is the parent of n_i. Then left context of n_i should appear in the child list of the merged node of n. If n_i has no left context, it should be the first node in the merged child list. Informally, a moved node should always have the same (non-deleted) predecessor and parent as it has in the branch it was moved in.

5. Assume $n_i \in T_1$ is a copied node and n is the parent of n_i. The node context of n_i should appear in the child list of the merged node of n. If the context is a left (right) context it should appear as first (last) in the merged child list. Informally, a copied node should always have the same (non-deleted) predecessor, successor and parent as it has in the branch it was copied in.

The propositions above state the structure in T_M near the areas of change. We will also need to state the structure of the tree in general. In the following, we state that the children of a node

$p \in T_M$ should either be the same as the children of the original node (in either branch), or a combination of the child lists of the nodes that were merged into p. A combined child list should only be made if the nodes merged into p are structural partners.

6. Assume $n \in T_1$ and n is not deleted. If n has no structural partner, its merged node p should have the children $p_1 \ldots p_k$ that are the merged nodes of the children $n_1 \ldots n_k$ of n. As the child list is taken directly from n, propositions 1–4 automatically hold.
   If n has a structural partner $m \in T_2$, its merged node should have the children $p_1 \ldots p_k$ that form the merged child list of n and m.

Propositions 2–4 states which subsequences should be available in the child list of a merged node, due to inserted, copied and moved nodes. We will now give a complete definition of the merged child list.

Merged child list Assume $n \in T_1$ and $m \in T_2$ are structural partners and $b \in T_B$ is their common base match. The merged child list of m and n is a list of nodes which contain all node subsequences implied by propositions 1–4 (partners $\mathbf{n}(i)$ and $\mathbf{m}(j)$ being equivalent). It also contains all child nodes of b that are not copied, moved or deleted in either branch. These nodes are in sequence with respect to $\mathbf{b}$. Furthermore, all nodes, except secondary copies, occur only once in the list. The nodes in the merged child list are the merge nodes of the respective nodes in $\mathbf{n}$ and $\mathbf{m}$.

Informally, we have stated that all moves and copies should be preserved in their context, and that unchanged nodes should also be present in the merged child list, in the same order as they were in the child list of the base node b. The equivalence condition states that there is no difference from which branch a node in a subsequence are taken, as long as the nodes are partners.

Note that all merging definitions and propositions are totally symmetric with respect to T_1 and T_2. Therefore the merge is also symmetric, as stated in requirement 7.

Finally, we will state the conflict situations in requirement 8 more precisely:

Update/Update conflict Assume $n \in T_1$ and $m \in T_2$ are content partners. If both nodes are updated with respect to T_B and their content cannot be merged, an update/update conflict occurs.

Delete conflict Assume $n \in T_1$ and $m \in T_2$ are partners, and that n is deleted with respect to T_B. A delete conflict occurs if m is updated, moved or copied.

Move/Move conflict A move/move conflict occurs for the node $b \in T_B$ if the following does not hold for b:

Let N be the set of nodes in T_1 that are moved and fully matched to b and M be the set of nodes in T_2 that are moved and fully matched to b. Let P be the set $N \times M$[1], i.e. all pairs of fully matched nodes to b. If P is not empty[2], there must exist a pair of nodes $(n, m) \in P$ whose parents x and y are structurally matched and and $n \in \mathbf{x}$ is in sequence with $\mathbf{y}$.

The definition seems quite complex, due to the fact that nodes may be copied. If copies were not allowed, and disregarding deletions, the set P would contain just one pair (n, m). The rest of the

[1] $N \times M$ is the Cartesian product of N and M.
[2] If P is empty, the node was deleted in either branch.

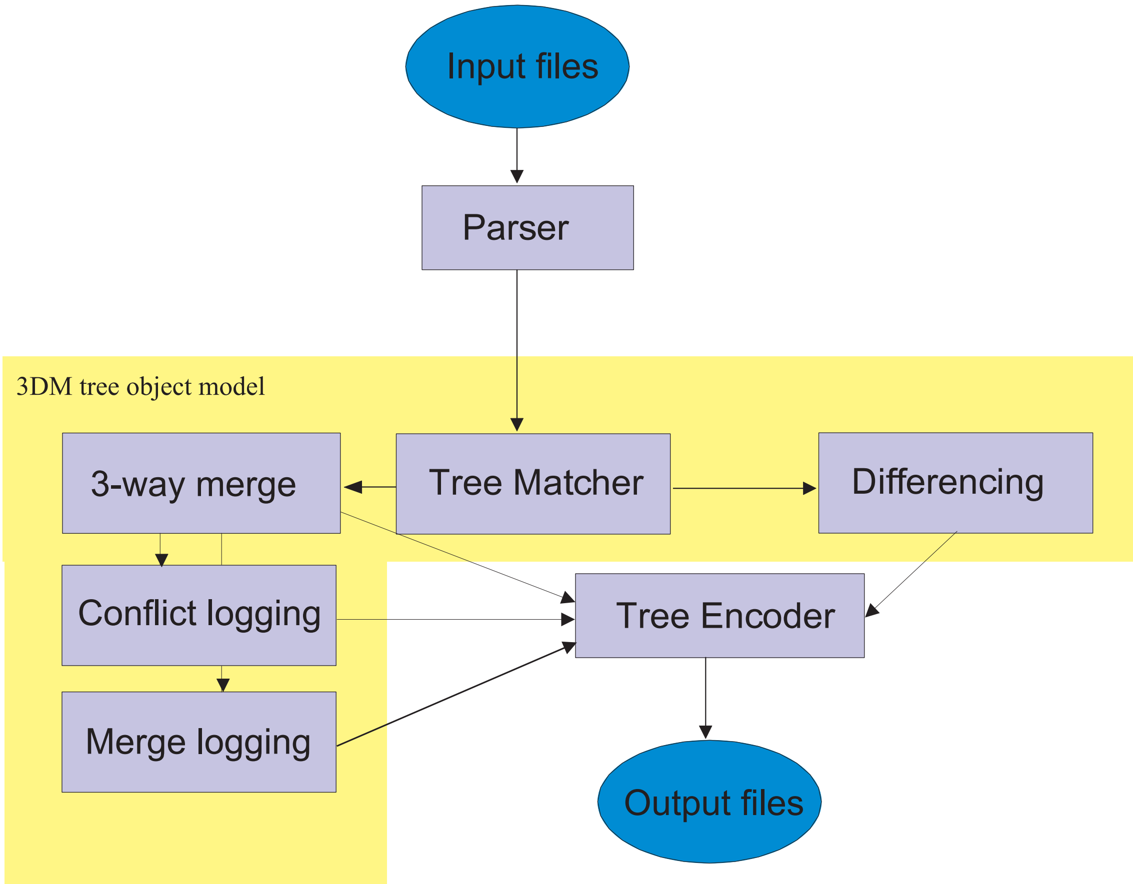


Figure 5.11: 3DM Architecture

definition simply states that both nodes in this pair must be moved to the same context of nodes: the same parents (the structural match criterion), and the same context in the child list (the in sequence criterion).

Finally, it is possible that the merged child list cannot be constructed. We'll call this a sequencing conflict.

Sequencing conflict Assume $n \in T_1$ and $m \in T_2$ are structural partners. A sequencing conflict occurs if the merged child list of n and m is a part of the merged tree, but no list can be constructed that satisfies the definition of a merged child list of m and n.

5.4 The architecture of 3DM

Figure 5.11 shows the overall architecture of 3DM. The tool reads its input form two or three XML files and parses the files to internal tree structures. The tree structures are then matched using the tree matcher. Using the tree matchings, either a 3-way merge of the trees or the difference between two trees is produced. The merging module also has facilities for logging a trace of edit operations and any conflicts that occur during the merge.

The output of the merge and differencing modules is also a tree structure, which is encoded to XML using the tree encoder. The resulting XML is then output to a file.

The tool makes no assumption about the data in the XML files. No modification timestamps or edit histories are required nor used. The elements need not be identified by unique IDs, but if they are, the tree matching can be made more reliable. The tool makes no assumption about the existence of an XML DTD (Document Type Definition) either. Currently the DTD is not used even if it exists; however, it could be used to improve the matching algorithm.

The architecture reflects the division of the tree merging problem into tree matching and merging of matched trees. The differencing functionality also benefits from this division, as we can directly utilize the tree matchings to produce the diff. This trinity, 3-way merging, Differencing and Matching of trees, forms the core modules of the merge tool and has also given the tool its name, "3DM".

The core modules will be presented in detail in the following three chapters. We start by presenting the tree matching algorithm in chapter 6, then present the 3-way merging algorithm in chapter 7, and finally the differencing algorithm in chapter 8. The parser and tree encoder is discussed in chapter 9.

The overall architecture was designed with the requirements of section 3.4 in mind. We will now review these requirements, and see how the requirements and the architecture fit together.

The tool interfaces with the outside world using standard XML files and the command line. In this respect, 3DM integrates with the UNIX toolbox philosophy of having a set tools that can be combined in almost infinite ways. Using files, as opposed to custom APIs or protocols, makes the tool highly interoperable with other applications.

As the tool supports the generation of diffs and operates on files, without requiring any additional knowledge of e.g. edit histories or synchronization events, it supports decoupled and weakly coupled environments very well. The file interface imposes no limitations on synchronization topology, and it can be used in a peer-to-peer as well as the more traditional client/server environment equally well.

The architecture is highly modular and flexible. For instance, supporting other tree-structured file types than XML is simply a matter of replacing the parser and tree encoder modules with those appropriate for the file type. The data exchanged between the merging, matching and differencing modules is limited to matched trees, and hence the modules show a high degree of independence, allowing them to be easily replaced or extended.

For instance, assume we want to make a merge tool utilizing edit histories. We could then replace the tree matching module with one that uses the edit history when building the matchings between the trees. If we were to design a 3-way merge tool for a particular file format, we could use a tree matcher that is aware of the semantics of the input data, e.g. by using the syntax rules in the DTD. Constructing an interactive merge would involve extending the merge module with a user interface.

5.4.1 Tracing the merge and producing readable diffs

As stated in the research problem in section 3.3 we want to be able to trace the operation of the merging algorithm, for purposes of presenting the user with information about changes in the merged document compared to the base document. Using the diff functionality of 3DM would be one possibility for producing such a trace, i.e. by running the tree differencing algorithm on T_B and T_M.

This approach has some shortcomings, though. No information about the origin of the change (i.e. which branch) can be provided using only the trees T_B and T_M. Furthermore, the differencing algorithm would have to be closely integrated with the merging algorithm to always interpret changes the same way the merging algorithm does.

Here, we will take a different approach. The merging algorithm will have a logging facility that logs the edit operations on the base induced by the branches, with emphasis on providing information useful for a human reviewing the changes. The difference algorithm will be constructed independently of the merging algorithm, with emphasis on a computer-friendly format for expressing the difference using an encoding that only contains the information necessary to reconstruct the target tree, not explicitly mentioning subtree copies, updates, deletes etc.

Expressing the diff between two files in a computer-friendly format naturally raises the question of how to produce a diff between two files that expresses the changes in terms of edit operations easily understood by human reviewers. Such a diff can be produced by the logging facility of the 3-way merging algorithm by letting one of the branches be identical to the base and the other be the target tree. For instance, to produce the human-friendly diff between two trees T_1 and T_2 we simply use the merge log of the 3-way merge of T_1, T_1 and T_2 (or T_1, T_2 and T_1).

Chapter 6

The Tree Matching Algorithm

Of the algorithms reviewed in section 4.4.1, only the heuristic matcher in [CG97b] allows a rich set of operations in moderate time. However, an expected time of $O(n^2)$ is still unacceptable for large data structures. Here we will use a simple heuristic matcher, with a worst-case running time of $O(n^2)$, which can be reduced to $O\left(n(e + \log n)\right)$, where e is the number of edits, by making some simple and reasonable assumptions about the trees. This matcher should be considered to be a proof-of-concept implementation of the matcher module, that should be replaced by a more well-designed matcher later on. Defining a truly good matching algorithm, that makes correct matches and is sufficiently fast is a research topic on its own, which will be left outside the scope of this thesis.

Although simple, the matching algorithm presented here has several desirable characteristics:

- It is reasonably fast
- Actual runs performed on the merge and use cases indicate that the matches generated are usually well-formed
- It illustrates how to implement a matcher with typed, i.e. content and structural, matchings.

It is possible to improve the quality of the matching if 3DM is used only for a certain type of data (e.g. XML with unique node identifiers, XHTML documents), or if an edit history is available. The reader should consider the matcher presented in this chapter to be a first attempt at a general-purpose matcher, which can be replaced by improved or domain-specific implementations as the need arises. An interesting alternative matcher would be the one presented in [CG97b] combined with the `setMatchTypes` postprocessing stage, presented in section 6.1.5.

6.1 A heuristic tree matching algorithm

The basic idea behind the heuristic matcher is simple: find as large as possible matching subtrees of T_B and T_1, and match the nodes according to the matching subtrees. Subtree matching is performed using a greedy algorithm, which takes a node in T_1 and the tree T_B as input, and returns a node in T_B which is the root of the largest matching subtree.

After the initial matching, some additional measures are taken to improve the quality of the matching and to determine the type of the matches. The postprocessing stages that improve the quality are: 1) matching unmatched nodes that are similar or in a similar context and 2) removing matchings that cause small amounts of data to be copied. Finally, the types of the matchings are determined. All in all, the algorithm consists of the steps shown in listing 1.

Listing 1 Top-level matching procedure

procedure buildMatching(T_B, T_1 : trees)
1: matchSubtrees(T_B, $T_1(R)$)
2: removeSmallCopies(T_B, T_1)
3: matchSimilarUnmatched(T_B, T_1)
4: setMatchTypes(T_B, T_1)
endproc

In the following sections we will look into each of these steps in detail. First we will, however, consider how to measure node similarity. A node similarity measure will come in handy when performing fuzzy node matching.

6.1.1 Measuring node similarity

In this section we will define two measures of node similarity: content similarity and child list similarity. These similarity measures are used by the matching algorithm for choosing a best match, when no exact matches exists.

The basis for both distances is the *q-gram string distance* defined in [Ukk92]. The q-gram distance was used instead of the more commonly used edit distance due to running time considerations. The q-gram distance can be computed in $O(n)$ [Ukk92], whereas the commonly used edit distance algorithm by [Mye86] requires $O(nd)$, where d is the edit distance. As several of the comparisons are made between totally unrelated strings, d is often close to n, yielding an average complexity close to $O(n^2)$ for calculating the edit distance using the latter algorithm.

Other methods include character frequency counting [CG97b] and calculating the difference between the character histograms, which is the same as the q-gram distance with $q = 1$, as well as vector similarity-based measures (see e.g. [MS99, chapter 15]). Character frequency counting suffers from the problem that the relative frequencies of letters in sufficiently large text corpuses in the same language are approximately the same, and hence any texts of sufficient equal length and same language will have almost identical letter histograms, and thus a very small distance.

[Ukk92] defines the q-gram string distance between two strings as follows:

Let Σ be a finite alphabet, Σ^* the set of all strings over Σ and Σ^q be all the strings of length q over Σ. A *q-gram* is a string $v = a_1 a_2 \dots a_q \in \Sigma^q$.

Let v be a q-gram and $x = a_1 a_2 \dots a_n$ a string in Σ^*. If $v = a_i a_{i+1} \dots a_{i+q-1} \subseteq x$ for some i, then v occurs in x. We denote the number of occurrences of v in x with $G_q(x)[v]$. The *q-gram profile* of x is the vector $G_q(x) = (G(x)[v])$, $v \in \Sigma^q$.

The *q-gram distance* between the strings x and y is the L_1-norm of $G_q(x) - G_q(y)$: $D_q(x, y) = \Sigma_{v \in \Sigma^q} |G_q(x)[v] - G_q(y)[v]|$.

For instance, assume $q = 2$ and $x = Linux$ and $y = Tux$. The 2-grams occurring in x are (Li, in, nu, ux) and those occurring in y are (Tu, ux). The q-gram profile for x is $G_2(x)[Li] = 1$, $G_2(x)[in] = 1$,$G_2(x)[nu] = 1$,$G_2(x)[ux] = 1$ and $G_2(x)[v] = 0$ for all other 2-grams. For y we have $G_2(y)[Tu] = 1$, $G_2(y)[ux] = 1$ and $G_2(y)[v] = 0$ for all other $v \in \Sigma^2$. Listing the 2-grams starting with Li, in, nu, ux, Tu (in that order) the 2-gram profile for x is $(1, 1, 1, 1, 0, 0, 0, \dots)$ and for y $(0, 0, 0, 1, 1, 0, 0, \dots)$. The distance $D_2(x, y)$ is thus 4 (out of maximally 5).

When calculating $D_q(x, y)$ in 3DM we conceptually[1] append a run of length $q - 1$ of a special character to x and y. The special character is selected so that it does not appear in x nor y. By doing this, we make $D_q(x, y)$ compatible with the editing distance, in the sense that both are between 0 and $length(x) + length(y)$. The upper bound occurs when x and y have no q-grams in common. Since the augmented strings x and y have exactly $length(x)$ and $length(y)$ q-grams, the upper bound is $length(x) + length(y)$. Using this method, the 2-grams in the previous example would be $(Li, in, nu, ux, x\cdot)$ and $(Tu, ux, x\cdot)$, where $\cdot$ is the special character. The distance would still be 4, as there's an equal amount of the 2-gram $x\cdot$in both strings. The maximal distance would be $8 = length(Linux) + length(Tux)$. To avoid the effect of similar q-gram profiles between unrelated, but sufficiently long texts, we increase the value of q with the combined length l of the strings, according to

$$q = \begin{cases} 1 & l < 50 \\ 2 & 50 \leq l < 150 \\ 4 & l \geq 150 \end{cases} \tag{6.1}$$

The constants 50 and 150 are approximate values. We denote this distance function by $\mathrm{qdist}(\cdot, \cdot)$.

The similarity measure between nodes is normalized to the interval $[0, 1]$, where 1 denotes maximal dissimilarity. We do not use the distance measure directly to measure the similarity, as we do not want matching strings that are short to be considered too good matches. This is because we use the similarity as measure of certainty that one node really does match another. Although the the distance is 0 between two nodes whose content is the same single letter, it does not make sense to assume that these must match, as the single letter apparently does not convey much information[2]. If the content of the nodes were two identical long sentences, we could be much more certain. To account for this phenomenon, we form the node similarity from the weighted average of the q-gram distance and a penalty term, where short content is assigned more penalty.

We define the *content distance* between two nodes to be

$$\mathrm{infoSize}(n) = \begin{cases} \max(|\mathrm{text}(n)| - c_t, 1) & , n \text{ is a text node} \\ c_e + \sum \mathrm{attr}_{(n,i)} \left(c_a + \max(|\mathrm{val}(n,a)| - c_v, 1)\right) & , n \text{ is an element node} \end{cases} \tag{6.2}$$

$$\mathrm{attrInfo}(a) = \min(|\mathrm{val}(n_1, a)| - c_v, 1) + \min(|\mathrm{val}(n_2, a)| - c_v, 1) \tag{6.3}$$

$$\mathrm{contentDist}(n_1, n_2) = \begin{cases} \min\left(\frac{\mathrm{infoSize}(n_1) + \mathrm{infoSize}(n_2)}{2}, \right. & n_1, n_2 \text{ text nodes} \\ \qquad \left. \mathrm{qdist}(\mathrm{text}(n_1), \mathrm{text}(n_2))\right) & \\ \mathrm{sametag}(n_1, n_2) + \sum_C \min(\mathrm{qdist}(\mathrm{val}(n_1, a), & n_1, n_2 \text{ element nodes} \\ \qquad \mathrm{val}(n_2, a)), \mathrm{attrInfo}(a)) + c_a D & \\ 1 & \text{otherwise} \end{cases} \tag{6.4}$$

where

[1] In reality, we let the string end with q-grams of length $1 \ldots q - 1$ in $G_q(x)$

[2] A more advanced model would take into account the frequency of a certain content when estimating the information content. That is, if a content occurs only once in the input trees, its information content is higher than if it is the content of every second node.

c_t,c_v are constants diminishing the contribution to the similarity of short texts and attribute values. For instance, an attribute value shorter than c_v will be treated as having a length of 1.

c_a is the information content in the attribute name

c_e is the information content of the element name

C are the set of attributes in both n_1 and n_2

D is the number of attributes from n_1 and n_2 not present in both nodes

sametag$(\cdot,\cdot)$ returns c_e if the element names are equal, otherwise 0

The functions infoSize$(\cdot,\cdot)$ and attrInfo$(\cdot,\cdot)$ return truncated length values for the length of the content and attribute values of a node. The purpose of these functions is to limit the contribution of short texts and attribute values to the similarity measure.

The maximum distance between two nodes is now

$$\text{maxDist}(n_1,n_2) = \begin{cases} \frac{\text{infoSize}(n_1)+\text{infoSize}(n_2)}{2} & n_1,n_2 \text{ text nodes} \\ c_t + \sum_C \text{attrInfo}(a) + c_a D & n_1,n_2 \text{ element nodes} \\ 1 & \text{otherwise} \end{cases} \tag{6.5}$$

We also define a penalty function for nodes containing little information as

$$\text{penalty}(n_1,n_2) = \max\left(1 - \frac{\text{maxDist}(n_1,n_2)}{c_p}, 0\right) \in [0,1] \tag{6.6}$$

where the penalty decreases linearly from 1 to 0 when maxDist$(\cdot,\cdot)$ increases from 0 to c_p. We can now define the node similarity as

$$\text{nodeDistance}(n_1,n_2) = \text{penalty}(n_1,n_2) + (1-\text{penalty}(n_1,n_2)) \cdot \frac{\text{contentDist}(n_1,n_2)}{\text{maxDist}(n_1,n_2)} \tag{6.7}$$

We also define the child list distance and the matched child list distance between two nodes. The former measures the similarity of the child lists of a pair of nodes using the content of the children, whereas the latter measures the similarity using the matching between children. The distance is based on turning the child lists into Unicode character strings, and then calculating the *qDist* function on these strings.

For a node n, we define the *content string* of the child list to be

$$\text{contentStr}(n) = a_1 a_2 \ldots a_N \tag{6.8}$$

where each character a_i is formed from the corresponding i:th child and $a_i = a_j \Leftrightarrow \mathit{content}(\mathbf{n}(i)) = \mathit{content}(\mathbf{n}(j))$. Similarly, we define the *match string* of the child list of a nodes n and m to be

$$\text{matchStr}(n) = a_1 a_2 \ldots a_N \tag{6.9}$$

$$\text{matchStr}(m) = b_1 b_2 \ldots b_N \tag{6.10}$$

where each character a_i and b_i is formed from the corresponding i:th child of n and m and $a_i = b_j \Leftrightarrow \mathit{matches}(a_i, b_j)$.

The child list distance and the matched child list distances are now

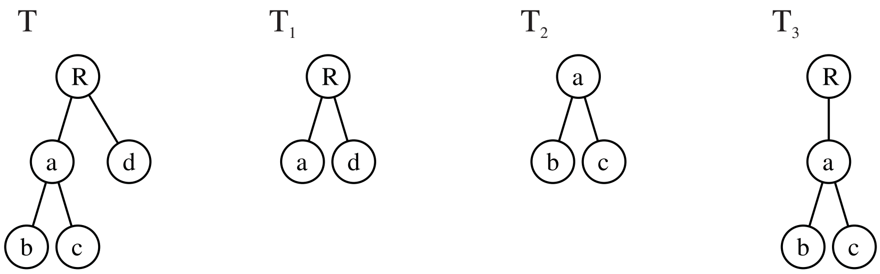


Figure 6.1: Truncated subtrees. The trees T_1 and T_2 are truncated subtrees of T, whereas T_3 is not, since the root does not have the same children as in T.

$$\text{childListDistance}(n_1, n_2) = \frac{\text{qdist}(\text{contentStr}(n_1), \text{contentStr}(n_2))}{\text{length}\,(\mathbf{n}_1) + \text{length}\,(\mathbf{n}_2)} \tag{6.11}$$

and

$$\text{matchedChildListDistance}(n_1, n_2) = \frac{\text{qdist}(\text{matchStr}(n_1), \text{matchStr}(n_2))}{\text{length}\,(\mathbf{n}_1) + \text{length}\,(\mathbf{n}_2)} \tag{6.12}$$

6.1.2 Subtree matching: `procedure matchSubtrees`

We begin the description of the subtree matcher by defining the concept of a *truncated subtree*. In short, a truncated subtree of a tree T is a subtree of T, which have all descendants of some of the nodes in the subtree removed. Figure 6.1 illustrates the concept of truncated subtrees.

Truncated subtree Let T be a tree, and T' a tree for which all nodes $n \in T'$: $n \in T$. Let T_s be the tree which is obtained from T' by replacing all leaf nodes in T' with the subtrees of T rooted at the leaf nodes. T' is a *truncated subtree* of T iff T_s is a subtree of T.

The `matchSubtrees` procedure works by recursively traversing T_1 and looking for matching truncated subtrees in T_B. Usually several matching subtrees are found, in which case the subtree with most nodes is used. In case there are several matches with the same maximal number of nodes, heuristics are used to determine the best match.

Once the best matching subtree has been selected, the subtree nodes in T_1 and T_B are matched. The subtree nodes in T_1 are also tagged with a subtree tag, which uniquely identifies the subtree. This is done to keep track of of the matched subtrees for later stages. Finally, the procedure recurses for each child of the leaf nodes of the matched subtree, i.e. the nodes that were not yet matched.

We will now look at each stage in some detail. The algorithm is presented in listing 2.

1. Finding matches
   When searching for matches for n in T_B, nodes whose content match exactly are considered first. If no such nodes are found, we return all nodes $m_i \in T_B$ for which $\mathit{nodeDistance}(n, m_i) \leq \min(c_1 \cdot \mathit{minDist},\, c_2)$, where $\mathit{minDist}$ is the minimum node distance between a node in T_B and n, c_1 is a constant indicating how much worse matches than the optimal we will consider and c_2 is a cutoff constant to prevent including too vague matches. Typically $c_1 = 2$ and $c_2 = 0.4$.

2. Finding matching subtrees
   Subtrees are matched by depth-first traversal, starting at n and m_i, and recurse as long as

Listing 2 The subtree matching procedure.

procedure matchSubtrees(T_B : Tree; n : node in T_1)
$(m_1, \ldots, m_i)$:= find matches for n in T_B
$\langle T_1, \ldots, T_i \rangle$:= find truncated subtrees rooted at m_i in T_B matching the truncated subtree rooted at n
$\langle T_{j_1}, \ldots, T_{j_k} \rangle$:= the subtrees with the maximal number of nodes
T := the best match of $\langle T_{j_1}, \ldots, T_{j_k} \rangle$
if the best match T exists **then**
  match the nodes in T with the corresponding nodes in T_1
  tag the subtree in T_1 with a unique subtree tag
end if
for each leaf node l in T **do**
  for each child l_i of l **do**
    matchSubtrees(T_B, l_i)
  end for
end for
endproc

the current nodes of the traversal has exactly the same number of children, and the content of the children match exactly. We call such a truncated subtree a *matching subtree.*

3. Selecting the best matching subtree
   This actually consists of two substeps: selecting the subtree with the maximum number of nodes, which is the greedy part of the algorithm, and if several subtrees have the maximum number of nodes, choose the best matching subtree among those. The latter step is described in more detail below. In case all matches are bad, no subtree is selected as best match.

4. Matching and recursing
   After the best matching truncated subtree has been selected, the nodes of the subtree are matched with the corresponding nodes in T_1. The nodes of the subtree in T_1 are also tagged. Finally, the procedure recurses.

Selecting the best matching subtree among equally large candidate subtrees

A common case leading to several matching subtrees where the maximum size is one node is the editing of a paragraph in an XHTML document. If the matching procedure, during the matching process, starts from a `<p>` tag, there will typically be a lot of exact matches for this node (since there are usually plenty of paragraph tags in an XHTML document). Since the text of the paragraph was edited, and assuming the editing was large enough to prevent fuzzy matching, no more nodes can be added to any of the matching subtrees, causing the size of the maximum match to be 1 node.

A solution that works quite well in these situations, where we have several candidates of the maximum size, is to look at the context of the node n for which we are searching for a match, and comparing it to the contexts of the roots m_i of the potential matches. The match with the most similar context is used. For instance, assume that the left sibling n_l of n was matched to m_l. If m_l is the left sibling of any of the potential matches, that match is likely to be the correct one. This is illustrated in figure 6.2.

Here, we have used the following method: Let n_l and n_r be the left and right siblings of n, and m_{l_i} and m_{r_i} the left and right siblings of each match candidate m_i. Assume that the node

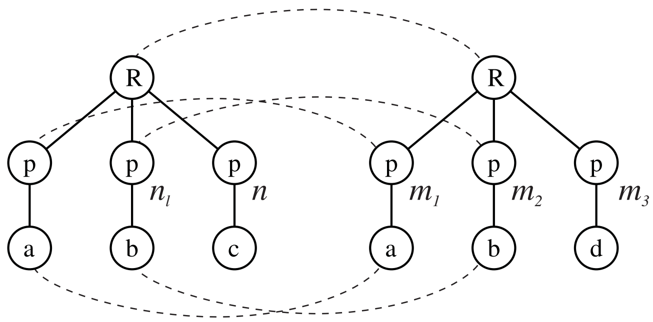


Figure 6.2: Finding the best match. The candidates for the node marked n are the nodes marked m_1, m_2 and m_3. We select the last p node (marked m_3), because its left sibling (marked with m_2) is matched to the left sibling of p, i.e. the node marked n_l.

distance between n_l (n_r) and m_{l_i} (m_{r_i}) is infinite if neither exist. The context distance measure between n and m_i is now

$$\begin{aligned} &\text{contextDistance}(n, m_i) = \qquad (6.13)\\ &\quad \min(\text{nodeDistance}(n_l, m_{l_i}), \text{nodeDistance}(n_r, m_{r_i}), \text{nodeDistance}(n, m_i)) \end{aligned}$$

If we find a candidate m_i for which n_l and m_{l_i} match we choose m_i. If no such candidate is found, the node with the smallest context distance is used, provided that is otherwise a sufficiently good match to n. Sufficiently good match means that if the size of the best candidate is only one node, the context distance needs to be less than some small constant (here we have used 0.1), otherwise there is no best match at all.

6.1.3 Removing small copies: `procedure removeSmallCopies`

The first postprocessing stage deals with putting a threshold on how much data must be duplicated in order to consider it to have been copied, instead of just inserted (see merging requirement 5 on page 46). For instance, consider a text document where each word is tagged separately. If the original text is "Susan and Ben should do the programming." and the modified version is "Susan and Ben should do the programming. Susan should do the GUI.", we would not want the second instance of the word "Susan" to be considered a copy of the first - especially if we merge with a second version, "Joe and Ben should do the programming." In that case the merge would be the undesirable "Joe and Ben should do the programming. Joe should do the GUI."

The `removeSmallCopies` procedure checks all nodes in T_1, whose base match has several matches in T_1 (i.e. the base match is copied). If the copied subtree of T_B to which the node belongs has an information content smaller than a certain threshold, the match is removed, and hence the node is considered inserted. The *information content* of a subtree is a measure of how much information a subtree contains. We have used the following heuristic formula giving the information content for a subtree:

$$\text{treeInfoContent}(T) = \sum_{n_i \in T} \text{infoSize}(n_i) + c_{ed} \cdot \text{numEdges}(T) \qquad (6.14)$$

The constant c_{ed} is the information content in an edge. Here we have used the value $c_{ed} = 4$.

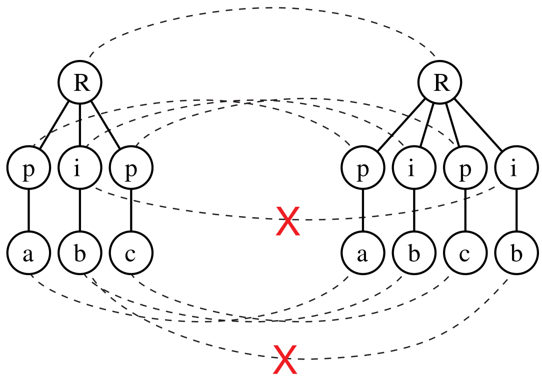


Figure 6.3: Determining which match to keep. The left and right siblings of the node i to the left match the left and right siblings of its base match, whereas those of the rightmost node i do not. Hence the match to the right i is removed, as well as the match of its child (since it is part of the same subtree). The removed matchings are marked with an X.

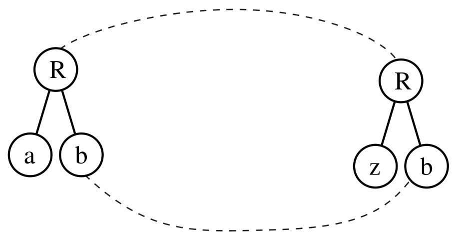


Figure 6.4: A matching that can be extended. The nodes a and z are not matched, although they most likely should. The procedure `matchSimilarUnmatched` handles such cases.

Recall from section 6.1.2 that all matched nodes in T_1 were tagged with a subtree tag to identify the matching subtree they are a part of. This tagging comes in very handy when implementing the `removeSmallCopies` procedure.

Sometimes, the subtrees of all matches are smaller than the copy threshold. In this case we do not want to remove all the matchings, but rather try to identify the best match, and leave it matched. We do it by looking at the neighbors of the match's subtree root: if nodes to the left and right match nodes to the left and right in the base tree, we add the info contents of the matching trees to that of the candidate for match removal. The candidate that gets most bytes (or more than the copy threshold) are not marked for unmatching. Figure 6.3 shows an example.

The pseudocode is in listing 3. The value of the copy threshold constant $c_{\text{threshold}}$ used by default was 128 bytes.

6.1.4 Matching similar unmatched nodes: `procedure matchSimilarUnmatched`

The second postprocessing stage of the matching algorithm tries to match unmatched nodes that are in a similar context, but were not matched by the subtree matcher, due to a too dissimilar content. For instance, consider the matching depicted in figure 6.4. Although the nodes a and z are very dissimilar it is very likely, judging from the matchings of the other nodes, that they should be matched. The task of `matchSimilarUnmatched` is to find such pairs of nodes and match them.

The rule for finding node pairs to match is very simple. Let m and n be a pair of unmatched nodes. If the parents of m and n are matched, and if either the left or right siblings (if they exist)

Listing 3 Procedure for removing "small" copies

procedure removeSmallCopies(m : node in T_B)
1: **if** m has several matches in T_1 **then**
2: $N :=$ the matches of m in T_1
3: **for** each $n_i \in N$ **do**
4: **if** tree info content of subtree n_i is tagged with $< c_{\text{threshold}}$ **then**
5: mark matching (m, n_i) for removal
6: **end if**
7: **end for**
8: **end if**
9: **if** all matches of m marked for removal **then**
10: $R :=$ the roots of the subtrees the nodes n_i are tagged with
11: **for** each $r \in R$ **do**
12: $s[r] := 0$
13: $r_b :=$ the base match of r
14: **while** r and r_b are matched and $s[r] \leq c_{\text{threshold}}$ **do**
15: add the treeInfoSize of r to $s[r]$
16: $r :=$ the left sibling of r
17: $r_b :=$ the left sibling of r_b
18: **end while**
19: do steps 13-18, but with right siblings
20: **end for**
21: unmark removal of (m, r) for the r whose $s[r]$ is largest
22: **end if**
23: **for** each child m_i of m **do**
24: removeSmallCopies(m_i)
25: **end for**
endproc

of m and n are matched, m and n should be matched. If neither siblings match (possibly because they do not exits), we will also match the nodes if both m and n are the last or first node in the child list. According to these rules a and z should be matched, as their parents and their right siblings, both b, match. If the b nodes didn't match, we would still match a and z as they are both the first node in the child lists.

We have to be somewhat careful when applying these rules, however. Consider the trees in figure 6.5. Although it is possible that the subtree rooted at b matches that rooted at x, it is not as likely as a matching z. Especially if the subtrees were larger without containing similar nodes at all, the likelihood of a match is quite low. If we were to apply the matching rules in a top-down fashion, the matches $b-x$, $e-y$ and $f-z$ would be created. This happens due to a "domino effect": first b and x are matched, leading to a parent and first-child match for e and y, a last-child match for f and z, and so on. There is no limit to the size (and decreasing likelihood of the matching) of the trees matched by the domino effect, and hence we want to avoid it. To avoid the domino effect we run the rules in a bottom-up fashion.

Another, more restrictive policy would be to mark all new matches created by the rules, and not consider them when looking for further pairs eligible for matching. That policy would disallow the domino effect in child lists, which may still occur when using the bottom-up approach. However, the bottom-up approach seems to work well in real-world situations. In the end, this is a question of how aggressively we want to match trees to one another. Some pairs of trees may have several completely disjoint areas (such as in use case 5), whereas others should only have a few disjoint nodes (see use case 4).

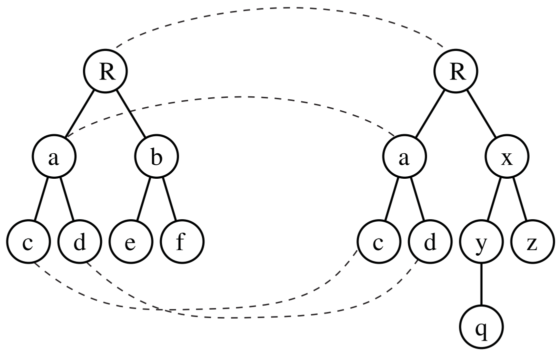


Figure 6.5: Partially matched trees. Should the subtrees rooted at b and x be matched?

6.1.5 Setting the types of the match: `procedure setMatchTypes`

So far, we have built the matching without considering if the matches are content, structural or full matches. The function of the last stage in the tree matching algorithm is to set the match type. The procedure visits each node b in the base tree T_B and checks what nodes in T_1 b matches. If b matches more than one node (i.e. it has been copied) we need to determine which of the copies is the original instance, whose match type is set to full match. After this, we set the match types of the other copies according to how they match b. If a copy is neither a structural nor a content match, the match between it and b is removed. Base nodes that have only one match in T_1 are naturally fully matched to that node.

In this implementation, the original instance is selected based on matched child list similarity between the base node b and the candidate n_i (the matched child list similarity is calculated as described in section 6.1.1). Should several nodes have a best matching child list, the node whose content is most similar to the content of b is used. Content similarity is measured using nodeDistance(b, n_i). The pseudocode for the `setMatchTypes` procedure is in listing 4. Since we use child lists matchings for determining the original instance, we need to run `setMatchTypes` on the children first, as it may remove matchings. Thus, the tree is traversed bottom-up.

Other possible approaches for identifying the original instance include looking at the largest matching subtree, in which there is a copy of b, and looking at the siblings and parent of the copies. The approach presented here was found to work well on the use cases.

6.2 Analysis

In this section we will analyze the complexity of the matching algorithm. As preliminaries to the analysis we will introduce two useful abstract data types (ADTs) and the complexities of the operations on these ADTs.

6.2.1 The set and map data structures

In the analysis of the matching as well as the merging algorithm we will frequently make use of two data structures: the set and the map. The set and map data structures both store items of arbitrary type, e.g. tree nodes or integers.

The set provides three operations: inserting an item to the set, querying if an item is in the set, and removing an item from the set. The map is similar to the set, with the addition that each

Listing 4 Procedure `setMatchTypes`

procedure setMatchTypes(b : node in T_B)
for each child b_i of b **do**
  setMatchTypes(b_i)
end for
if b has only one match n in T_1 **then**
  set match type of (b, n) to full
else
  $N :=$ nodes that match b
  find the node $n \in N$ that has the smallest $matchedChildListDistance(n.b)$
  if several nodes exist in N **then**
    $n :=$ the node with minimum node distance to b
    for each node $n_i \in N \setminus n$ do **do**
      if n_i does not match b (neither content nor structurally) **then**
        remove match (b, n_i)
      else
        set match type according to how b matches n_i, i.e. if they content match, the match type is content. If they do not match at all, remove the matching (b, n_i)
      end if
    end for
  end if
end if
endproc

item, now called the *key*, in the map has an associated value. The operations provided by the map are: inserting a key with associated value, querying the value for a given key and removing a key and its value.

Maps and sets can be implemented using Red-Black trees. Examples of such implementations are the `java.util.treemap`, `java.util.treeset` classes in Java 2 [Sun]. Implementations of the set and map using Red-Black trees have the property that the worst-case time complexity for all the above mentioned operations is $O(\log n)$, where n is the number of items in the set or map [GS78].

The set and map are also frequently implemented using hashing techniques. These implementations are typically faster, but the worst-case time complexity for the operations is worse than $O(\log n)$, and are thus not suitable for a worst-case analysis. The sets and maps used in the matching and merging complexity analyses are assumed to be implemented using Red-Black trees (or any other technique yielding $O(\log n)$ complexity on all operations), but in practice sets and maps based on hashing techniques were used in the implementation of 3DM.

6.2.2 Complexity analysis of the matching algorithm

We concluded the presentation of the matching algorithm by analyzing its complexity. Let the number of truncated subtrees and unmatched nodes that make up T_1 be D. Each of the subtrees contains m_i, $i \in [1, D]$ nodes; for unmatched nodes $m_i = 1$. A typical situation is depicted in figure 6.6.

For the sake of the analysis we divide the matching algorithm in two phases: node distance calculations and the actual matching. In reality the matching and distance calculations are intertwined. We well also assume that the matches of a node in T_1 is stored at the node, so the set of matches can be accessed in $O(1)$.

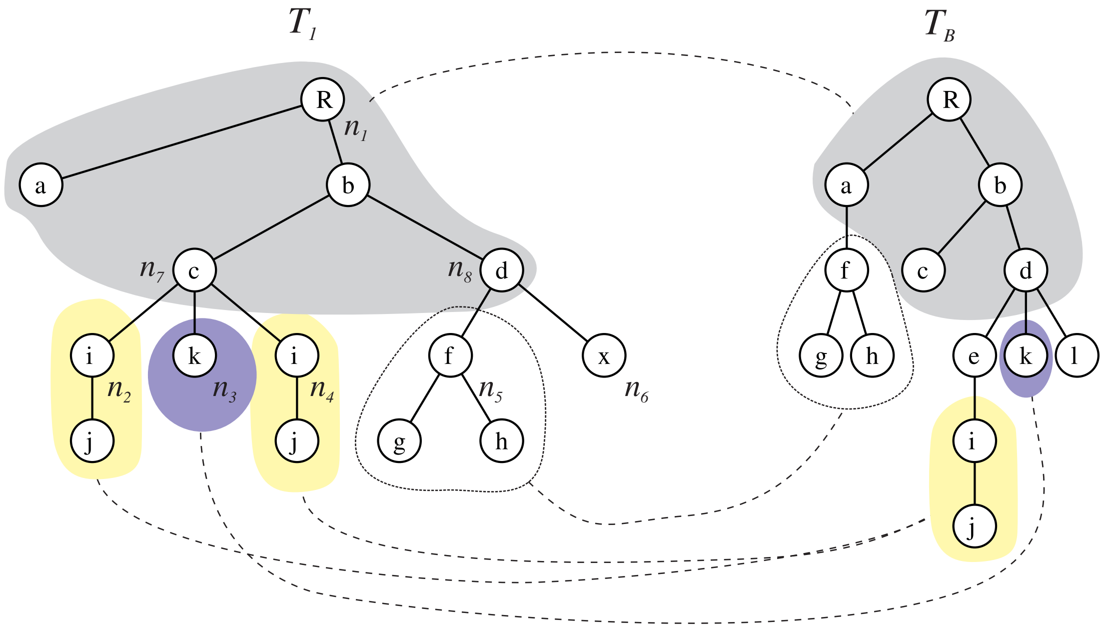


Figure 6.6: T_1 is made up of truncated subtrees matching subtrees in T_B. In the figure $D = 6$ and $D_p = 2$, since there is a total of six subtree roots and unmatched nodes in T_1, and there are two parents for the subtree roots and the unmatched nodes. The nodes $n_{j_1} \ldots n_{j_6} = n_1 \ldots n_6$ and $n_{j_{D+1}} \ldots n_{j_{D+D_p}} = n_7 \ldots n_8$.

Let the nodes of T_1 be $n_1 \ldots n_k$ and N the total number of nodes in T_1 and T_B combined. Let $n_{j_1} \ldots n_{j_D}$ be the nodes that will be the roots of matching subtrees and unmatched (i.e. inserted) nodes and $n_{j_{D+1}} \ldots n_{j_{D+D_p}}$ be the parents of $n_{j_1} \ldots n_{j_D}$. The roots of the matching subtrees are nodes that have siblings that do not have matches in T_1 or nodes that are out of order with respect to the base. That is, the roots are the children of the leaf nodes of the matched truncated subtrees (see section 6.1.2). The nodes $n_{j_1} \ldots n_{j_D}$ and $n_{j_{D+1}} \ldots n_{j_{D+D_p}}$ are illustrated in figure 6.6.

Furthermore, let N_c be the size of the content of the nodes and the child lists in T_1 and T_B combined and let e_i be the size of the content and child lists of the nodes n_{j_i}, $i \in [1, D + D_p]$. We can now use $E = \sum_i e_i$ to measure the total size of updated and inserted nodes and their child lists in T_1. Typically E is smaller than N_c, i.e. only a fraction of the nodes in T_1 have been edited.

We will also analyze the complexity under the assumption that the tree does not contain redundant data. We express this assumption in the following manner:

Assumption 1 When given the c_u first nodes visited by a depth first traversal starting at a node n in T_B the node n can be unambiguously identified.

Informally this means that when given a subtree of c_u nodes from T_B, we can unambiguously identify where in T_B the subtree is rooted, because there are no other identical subtrees in T_B.

Towards the end of the analysis we want to tie together the content size N_c and the number of nodes N. If we assume that each node has a maximal content size c_{max} then $N_c \leq c_{max}N$. Stating it differently, if a tree grows in size it is because more nodes are added, not because the content of the nodes is expanded. Although not true for every edit cycle, on average this should be correct.

Assumption 2 Each node has a maximal content size c_{max} , and hence $N_c \leq c_{max}N$.

We start the analysis by deriving an upper bound on the complexity for the node distance calculations.

Lemma 1 A table containing the similarity between all node pairs (n_{j_i}, n_l) where $n_{j_i} \in T_1$, $n_l \in T_B$, $i \in [1, D + D_p]$ and $l \in [1, k]$ can be computed in $O(N_c E)$

Proof As shown in [Ukk92] the q-gram distance can be computed in linear time. We obtain the following upper bound for the time complexity for constructing a table of node similarities, with only those entries required for the matching algorithm:

$$\begin{aligned}
\sum_{i=1}^{D+D_p} \left(\sum_{l=1}^{k} O(|n_l| + e_i) \right) &\leq \sum_{i=1}^{D+D_p} O(ke_i + N_c) \\
&\leq \sum_{i=1}^{D+D_p} O(N_c e_i) \\
&\leq O(N_c E) \qquad (6.15)
\end{aligned}$$

Lemma 2 A table of the childListDistance and the matchedChildListDistance (see equations 6.11 and 6.12 on page 63) between all node pairs (n_{j_i}, n_l), $n_{j_i} \in T_1$, $n_l \in T_B$, $i \in [1, D + D_p]$ and $l \in [1, k]$ can be computed in $O(N_c E)$

Proof Calculating the content and match strings of the child lists can be done in linear time, and the strings can be stored in space less than E, since the size of the child lists of n_{j_i} are included in E. We thus need to calculate the q-gram distance between a part of the e_i:s in E, and all of the content in T_B, denoted by N_c. The strings can be constructed in $O(E)$ and, comparing to lemma 1, a $O(N_c E)$ upper bound trivially follows on the distance calculation, for a total complexity of $O(N_c E)$.

Next, we analyze the complexity of the `matchSubtrees` procedure.

Lemma 3 The `matchSubtrees` procedure (excluding the recursive step) is $O(Ns)$, where s is the size of the subtree matched, or $O(N)$ if we take assumption 1.

Proof By analyzing each step of `matchSubtrees` we obtain:

- Step 1: We compare n to each node in T_B. Equality can be determined in $O(1)$, and so can nodeDistance$(n, \cdot)$, as we can use the table of node similarities, since n is the root of a matched subtree or an inserted node. The total complexity of the step is thus $O(N)$, and an upper bound on the number of matches found is N.
- Step 2: Let s_i be the size of the subtree T_i and $s = \max(s_i)$. For each subtree matching takes $O(s_i)$ time (child lists can be compared for equality in $O(1)$ using a hash value of the child list), for a total of less than $O(Ns)$. If we take assumption 1 the subtree is uniquely identified by the c_u first nodes in a depth-first traversal and the complexity becomes $O(c_u N + s) = O(N)$ (since $s \leq N$).
- Step 3: $O(N)$.
- Step 4: Calculating the contextDistance$(\cdot, \cdot)$ for a candidate can be done in $O(1)$, using the similarity table, since all the necessary entries are in the table. Obviously n is the root of a subtree and thus $n = n_{j_i}$ for some i. According to definition of which the subtree roots are, the left and right siblings of n must also be subtree roots, and thus in the similarity table. The same holds for the child list of n, its distance to every child list in T_B is also tabulated.

Evaluating the contextDistance$(\cdot,\cdot)$ function for maximally N candidates thus takes $O(N)$ time, including the scan for the best candidate.

- Step 5: This is a simple check that the node similarities are sufficiently good, $O(1)$
- Step 6,7: Since each node in the subtree needs to be matched and tagged, these steps take $O(s)$
- Steps 9-13 form the recursive step, and are not included.

Summing over the steps we get $O(Ns)$ or, according to assumption 1, $O(N)$.

As the procedure is called once for each matched subtree, it is called in total D times. The recursive step (steps 9-13) is $O(1)$ for at total of $O(D)$ time. The total complexity for the subtree matching becomes

$$O(D) + \sum_{i=1}^{D} O(Ns_i) = O(N^2) \tag{6.16}$$

where s_i is the size of each matched subtree, and hence $\sum s_i \leq N$.
In case we make assumption 1 we get

$$O(D) + \sum_{i=1}^{D} O(N_B) = O(N_B D) \tag{6.17}$$

Next, we analyze the complexity of the procedures `matchSimilarUnmatched`.

Lemma 4 The time complexity of `matchSimilarUnmatched` is $O(N)$

Proof Trivial. Each node in T_1 is visited once, and each visit requires a constant amount of time.

Lemma 5 The time complexity of `removeSmallCopies` is $O(N \log N)$

Proof Every node in T_B and, at most, every node in T_1 is visited in constant time. The information content for all subtrees can be calculated and tabulated in $O(N)$. Removing a matching takes at most $\log N$, using the set data structure for storing matches. If all matches are marked for removal, we must determine which matches to keep. This can also be accomplished in constant time for each match, as the while loop of steps 14-18 executes maximally $c_{\text{threshold}}$ times, since each each matched sibling adds at least 1 byte of information (see equation 6.2 on page 61).

The total complexity for visiting all nodes in T_B, and processing all copies (of which there are no more than T_1) in $O(1) + O(\log N)$ time becomes

$$O(N) + N \cdot O(\log N) = O(N \log N) \tag{6.18}$$

Lemma 6 The time complexity of `procedure setMatchTypes` is $O(N \log N)$

Proof The steps of `setMatchTypes`, excluding the recursion, take time:

- Step 4-5: $O(1)$
- Step 7: $O(1)$ since the matches are tabulated at the node.

- Step 8: Let m_i be the number of nodes that match b. Selecting the node that has the smallest matched child list distance can be done in $O(m_i)$, as we can use the table of child list distances and node similarities to calculate the distances for each node in $O(1)$. The required distances are tabulated, according to the following reasoning: The match in T_1 is either the root, the leaf of or an interior node of a matching subtree. In the former case the node is either some n_{j_i} or a parent of some n_{j_i}, both of which are cases when the distances are tabulated (see lemmas 1 and 2). If the match is an interior node (detected by its absence from the distance table), its child list as well as its content matches b exactly, according to the definition of matching subtrees, and the distances are 0.
- Steps 11-17: Determining the match type is done using equality comparisons for the content and child lists, both of which are done in $O(1)$ using hash codes. Removing a matching takes at most $\log m_i$ using the set data structure. As the loop executes m_i times, the complexity of the loop becomes $O(m_i \log m_i)$.

Summing over the recursive calls to `setMatchTypes`, of which there are $|T_B|$ yields

$$\sum_{i=1}^{|T_B|} O(m_i \log m_i) \leq O(N \log N) \tag{6.19}$$

since $\sum m_i \leq N$.

The total complexity of the heuristic tree matching procedure `buildMatching` now becomes, using lemmas 1, 2, 4, 5, 6 and equation 6.16:

$$2 \cdot O(N_c E) + O(N^2) + O(N) + O(N \log N) + O(N \log N) = O(N^2) + O(N_c E)$$

Under assumption 1 the complexity is (using equation 6.17)

$$2 \cdot O(N_c E) + O(ND) + O(N) + O(N \log N) + O(N \log N) = O\left(N(D + \log N)\right) + O(N_c E) \tag{6.20}$$

Finally, we make some observations regarding the relation between the number of edit operations e and D. A single edit operation will cause D to be the size of the affected child list plus one (the parent is included). A subsequent edit to the same child list will not, however, increase the value of D. By scattering the edits, the value of D is thus maximized. It seems reasonable, however, that edits are usually fairly concentrated, in which case we may think that $D \approx c_d e$, where c_d is some constant.

Under this observation as well as assumptions 1 and 2 we get the following theorem:

Theorem 1 The complexity of the matching algorithm under the assumptions 1, 2 and $D \propto e$ is $O\left(Ne + N \log N\right)$, where N is the number of nodes in T_B and T_1 combined, and e is the number of edits.

Proof Follows directly from equation 6.20 and the assumptions:

$$O\left(N(e + \log N)\right) + O(Ne) = O\left(N(e + \log N)\right) \tag{6.21}$$

The worst case time complexity for the matching algorithm is thus $O(N^2)$, which can be reduced to $O(N(e + \log N))$ using reasonable, non-restrictive assumptions on the structure of the trees.

Chapter 7

The Tree Merging Algorithm

We are now ready to present the main contribution of this thesis: a 3-way merge algorithm for ordered tree structures. The algorithm itself is presented in section 7.1. We then proceed with an analysis (section 7.2), by which we show some important properties of the algorithm, and analyze its complexity.

7.1 The merging algorithm

Unlike existing structural merging algorithms our merging algorithm directly uses the matchings $\mathcal{M}_{B1}$ and $\mathcal{M}_{B2}$, produced in the tree matching stage, to produce the merged tree. As no edit script is generated, there are no restrictions imposed by edit script operations. Neither is there a danger of strange behavior due to minimal edit script peculiarities, as described in [CG97b]. The only restriction imposed is that the matchings are natural, the most notable implication of which is that the uncopy operation is not allowed.

The main idea of the algorithm is to traverse the trees T_1 and T_2 concurrently so that partner nodes are visited simultaneously. Each step of the traversal outputs a node to T_M. The outputted node is the merge of the nodes in T_1 and T_2 currently being visited.

To aid the presentation of the algorithm we introduce the concept of a *tree cursor*. The tree cursor indicates the current position in a tree in much the same way as the cursor in a text editor indicates the current position in the text. The tree cursor may also be positioned at a special null position, indicating that the cursor is not active. We will denote cursors by C_n, where n is the name of the cursor, and refer to the node the cursor is pointing to with node(C_n). The null node is denoted by $\emptyset$. The notation for pointing a cursor at a node m is $C_n = m$. Let C_1, C_2 and C_M be the cursors of the trees T_1, T_2 and T_M respectively.

Partner nodes paired up for merging by the algorithm will be referred to as a *merge pair*. Merge pairs are denoted by $\{n, m\}$, where n a m are the nodes in the pair. Merge pairs may also contain only one node, in which case the pair is denoted by $\{n, \cdot\}$. The *merge pair list* is the list of merge pairs generated by combining the child lists of the nodes pointed to by C_1 and C_2.

Assume that the trees T_1 and T_2 are non-empty[1], that the matchings $\mathcal{M}_{B1}$ and $\mathcal{M}_{B2}$ exist and that they are natural, and that $T_M = R$. All cursors are positioned at the root of their associated tree. These are the initial conditions for the algorithm. The main steps of the algorithm are shown in listing 5.

[1] At least the root exists

Listing 5 The main procedure of the merging algorithm

```
procedure treemerge
  generate the merge pair list of the children of node(C1) and node(C2)
  p := node(CM)
  for each pair :={ui, vj} in the merged pair list do
     merge the contents of ui and vj into a new node w
     add w as a child of p, and reposition CM at w
     reposition C1 to u and C2 to v (with some exceptions)
     call treemerge
  end for
endproc
```

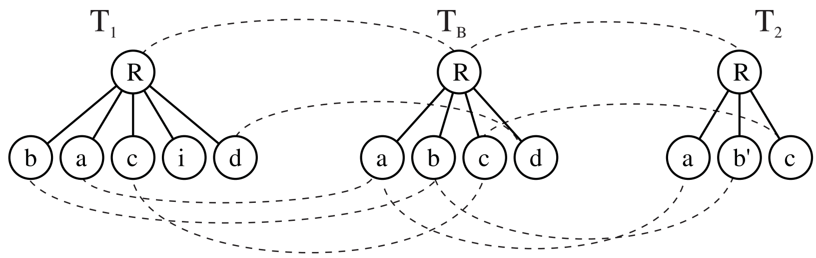


Figure 7.1: A simple merge example.

Before going into the details of each step, we will now explain the function of each step briefly and show a simple example of tree merging.

In the explanation below of the steps, we assume T_1, T_2 and T_B are the trees depicted in figure 7.1, that they are matched according to the figure, and that all cursors are positioned at the root of their tree.

1. Generate merge pair list

   In this step we pair up the children of node(C_1) and node(C_2) according to the matchings. We then determine the sequence of these pairs, according to any moves made in either tree. Deleted nodes are removed from the list. In this case, we pair up the child lists $b\,a\,c\,i\,d$ (of $T_1(R)$) and $a\,b'\,c$ (of $T_2(R)$). The resulting pair list would be

$$\begin{array}{cccc} b & a & c & i \\ b' & a & c & \cdot \end{array}$$

   The upper row contains nodes from T_1 and the lower from T_2. Note that the ordering of the pairs obeys the moves made in T_1 with respect to T_B, that d which was deleted in T_2 is not present, and that the inserted node i has no pair.

2. Merge the contents

   We now determine what the merged content of the pair $\{u_i, v_j\}$ should be. Generally speaking, we always pick the content which represents a change with respect to T_B. In this case the merged contents would be $b'\,a\,c\,i$.

3. Add w

   We have now successfully merged the nodes u_i and v_j into w. The merged node is added to the merge tree and the cursor position is updated to append children to the new node. By adding $b'\,a\,c\,i$ our merged tree is now $T_M = (R;\, b'\,a\,c\,i)$.

4. Reposition C_1 and C_2

   The cursors for T_1 and T_2 are updated to point to the nodes in the merge pair, whose content merge was added to T_M. Usually the cursors are pointed directly to u_i and v_j, but there are exceptions, occurring for instance when the pair has only one node, or u_i and v_j are not structural partners. The cursor updates for the merge pairs in the sample case are

$$\begin{aligned} \{b, b'\} &\Rightarrow C_1 = T_1(b) \wedge C_2 = T_2(b') \\ \{a, a\} &\Rightarrow C_1 = T_1(a) \wedge C_2 = T_2(a) \\ \{c, c\} &\Rightarrow C_1 = T_1(c) \wedge C_2 = T_2(c) \\ \{i, \cdot\} &\Rightarrow C_1 = T_1(i) \wedge C_2 = \emptyset \end{aligned}$$

5. Call `treemerge`
   This is the recursive step of the algorithm. Note that we have now successfully added a merged node of the children of the nodes pointed to by the original values of C_1 and C_2 to T_M . The cursors C_1, C_2 have been reset to point to a new set of partners (corresponding to the merged child), and C_M is updated to the new "insert" position in T_M.

In the following sections, we will have a detailed look at steps 1, 4 and 6. As an overview, the steps of the merge algorithm for a simple merging example is illustrated in figure 7.2.

7.1.1 Generating the merge pair list

Let $u = \text{node}(C_1)$ and $v = \text{node}(C_2)$. In this step we will create a list of node pairs to merge from the child lists of u and v. The order of the pairs in the list will obey all move, copy, insert and delete operations made in T_1 and T_2. As an example we will show how to generate the merge pair list when merging the root nodes of the trees depicted in figure 7.3.

First, we generate *merge lists* from the child lists of u and v. Then we check for deletions and far moves in either list, and process these. Finally, we combine the merge lists into the merge pair list, simultaneously resolving the order of the pairs.

If only one of the cursors point to a node (i.e. the other cursor is at the null position) we do not need to merge any child lists, and can thus generate the merge pair list directly from the child list of the active cursor. We start by presenting the more difficult case of merging two child lists, and describe how to generate the merge pair directly from the child list at the end of the section.

Generating the merge list

The purpose of the merging list is to transform the child lists into a suitable representation for the combine and reordering phase. In the merge list moved, inserted and copied nodes are locked into their context, either by being marked as locked or added as a *hangon* to another node in the merge list.

The merge list starts with the special start entry α, followed by any number of entries, and ends with the end entry ω. Each entry contains a node, a flag indicating if the node is locked, and any number of hangon nodes. Merge lists are written like this:

$$M_1 = \begin{array}{ccccc} i & & j & & \\ \underline{\alpha} & \underline{b} & \underline{c} & \underline{d} & \omega \end{array} \tag{7.1}$$

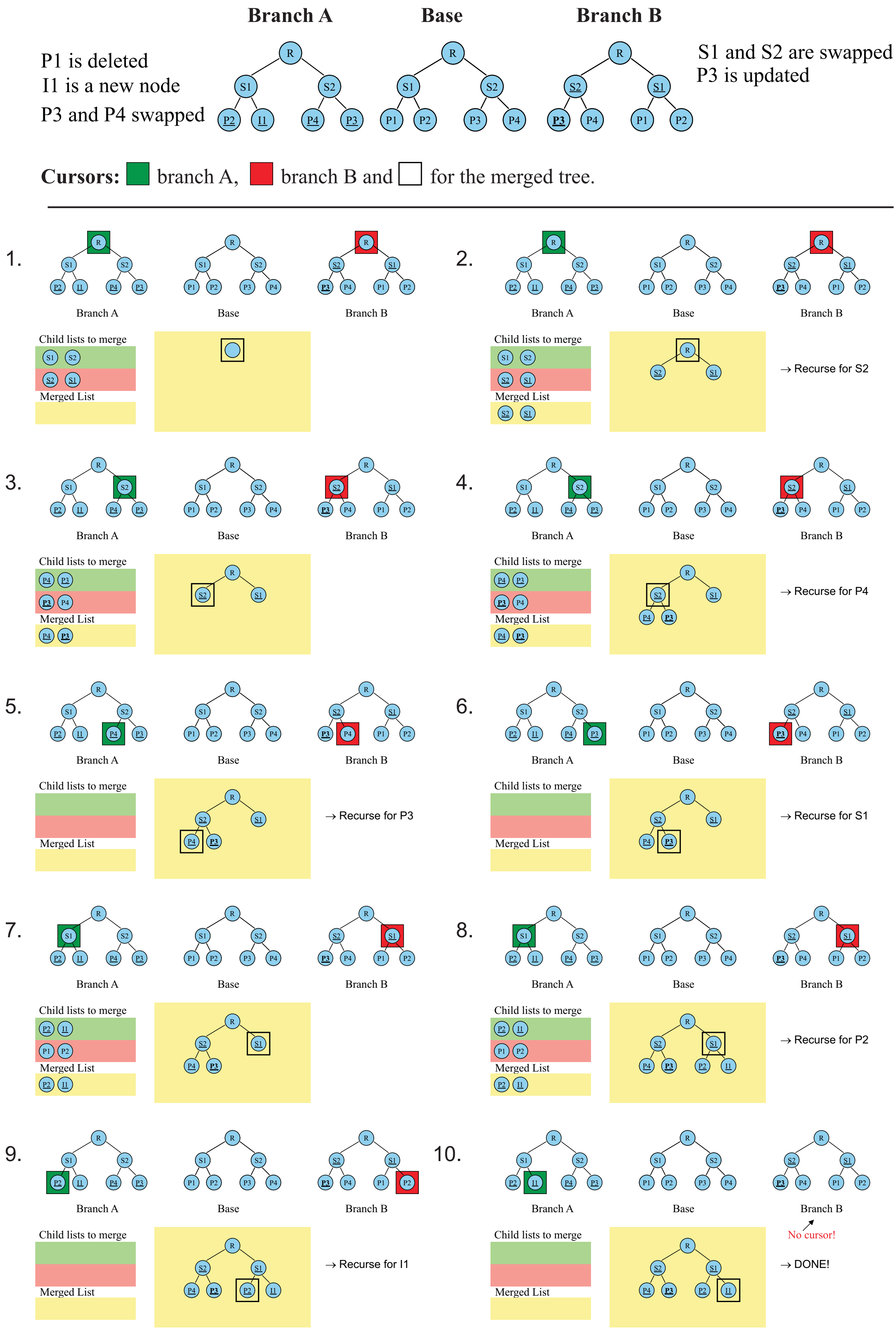


Figure 7.2: Illustration of the steps of the merge algorithm for a simple merging example.

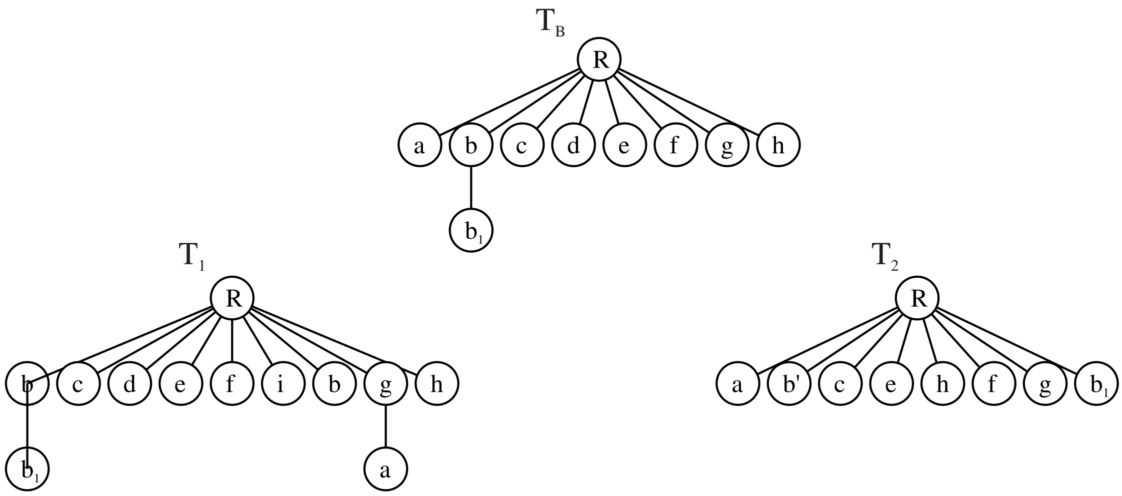


Figure 7.3: Sample case used to illustrate the generation of the merge pair list. Nodes with the same label are matched, as well as $T_B(b)$ and $T_2(b')$.

Note that hangon nodes are shown as stacked on top of the node they belong to. We refer to entries (excluding the start and end entry) by the label of their associated node.

In the merge list M_1 above the nodes i and j are hangon nodes. The entries α, b, c and d are locked. Note that the start and end entries may be locked, and that the start entry may have hangons.

We say that an locked entry $m \in M$ is *left locked* if its preceding entry is locked, and *right locked* if its succeeding entry is locked.

The semantics of the merge list are as follows:

- If two consecutive entries (excluding the start and end entries) are locked, the sequence $m_1\, i_1 \dots i_n\, m_2$ must appear in the merged child list, where m_1 and m_2 are the locked entries, and $i_1 \dots i_n$ are the hangons bottom-to-top of m_1.
- If the start entry α is right locked, the merged child list must start with $i_1 \dots i_n\, m$, where $i_1 \dots i_n$ are the hangons of α and m is the entry succeeding α.
- If the end entry ω is left locked the last nodes in the merged child list must be $m\, i_1 \dots i_n$, where m is the entry preceding ω and $i_1 \dots i_n$ are its hangons.
- If a entry m has hangons (bottom-to-top) $i_1 \dots i_n$ the sequence $m\, i_1 \dots i_n$ must appear in the merged child list.

For instance, given the merge list

$$M_2 = \begin{array}{ccccccc} & & & j & & & \\ & & & i & & & \\ \underline{\alpha \quad b} & & c & \underline{d \quad e \quad f} & & & \omega \end{array} \tag{7.2}$$

the merged child list must start with b and contain the subsequence $d\, i\, j\, e\, f$. The entries in the merge list are indexed starting from zero (which is α), the n:th entry being denoted by $M(n-1)$. For instance, in the list above $M_2(3)$ is d.

When generating the merge list from the child list of a node n we want children that are secondary copies in **n**, or not present in the child list of the base node b of n to be added as hangons, with the entry they are hanged on to right locked. Children of n not in sequence with respect to **b** are added as a left locked entries. Primary copies are locked both to the left and right if secondary copies exist in **n**. This way, locking and hangons in the merge list reflect the required subsequences according to the formalized merging requirements 3, 4 and 5. The usefulness of the distinction between locked node sequences and hangons, as well as the distinction between primary and secondary copies, will become evident when building the merge pair list from the merge lists. The pseudocode procedure generating the merge list is presented in listing 6.

Listing 6 Pseudocode for generating the merge list

```
procedure makeMergeList( n: Node)
M :=empty merge list
append α to M
let b = the match of n in T_B
let b = the child list of b.
for each child n_i of n do
   if n_i is inserted then
      add n_i as hangon to the last entry in M
      mark the last entry to be left locked
   else if n_i is far moved then
      add n_i as hangon to the last entry in M
      mark the last entry to be left locked
   else if n_i is a secondary copy of a then
      add n_i as hangon to the last entry in M
      lock the primary copy of a to the left and right
   else
      append to M a new entry e whose node is n_i
      if n_i is not in sequence with b then
         lock e to the left
      end if
   end if
end for
let n_i be the node of the last entry in M
append the end entry ω to M
if the match of n_i is not last in b then
   lock ω to the left
end if
endproc
```

The merge lists for the nodes $T_1(R)$ and $T_2(R)$, denoted by M_1 and M_2, (the base node being $T_B(R)$ in both cases) from the merging example in figure 7.3 are

$$M_1 = \begin{array}{ccccccccc} & & & & & b & & & \\ & & & & & i & & & \\ \underline{\alpha} & \underline{b} & \underline{c} & d & e & \underline{f} & \underline{g} & h & \omega \end{array} \tag{7.3}$$

$$M_2 = \begin{array}{ccccccccc} & & & & & & & b_1 & \\ \alpha & a & b' & c & \underline{e} & \underline{h} & \underline{f} & \underline{g} & \underline{\omega} \end{array} \tag{7.4}$$

In M_1 the sequence $\alpha\, b\, c$ is locked because the b in the sequence is the primary copy of b (the secondary copy being the b inserted after i) and f is locked to g due to the inserted node i as

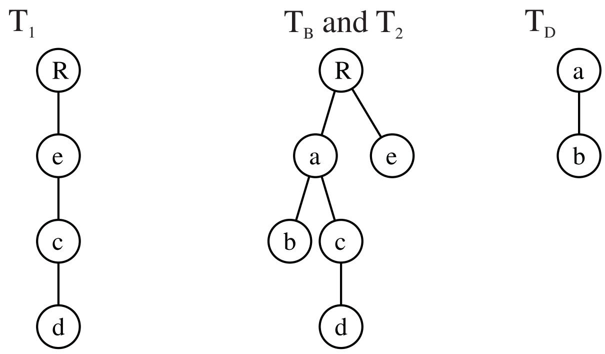


Figure 7.4: Delete Tree. The delete tree of $T_2(a)$ that needs to be checked for edits when merging the child lists of $T_1(R)$ and $T_2(R)$ is the tree T_D in the figure.

well as the secondary copy of b. The nodes i is a hangon because it is inserted, and b is a hangon because it is a secondary copy.

In M_2 h is left locked to e because h is not in sequence with the base child list. For the same reason, f is left locked to h. The node g is locked to ω because it has a hangon (b_1), and g was not the last node in the original child list. As b_1 is far moved from below b, it is added as a hangon to g. Note that c is not locked to e, as these nodes are in sequence. This is because the node in between, d, was deleted.

Checking for deletions and far moves

Let the merge lists generated from the children of u and v be M_1 and M_2. The common base node of u and v is b. We now start combining the merge lists by checking for nodes present in $\mathbf{b}$ but not in M_1 or M_2, i.e. nodes that have been deleted or far moved away from $\mathbf{b}$ in either branch. In short, any node $b_i \in \mathbf{b}$ not present as the node of an entry in M_1 that has an entry match (i.e. the node of an entry is matched to b_i) in M_2 has its entry removed from M_2, and vice versa.

Unfortunately, we cannot just remove entries from either merge list without some care. We need to check that there are no edits in the tree of deleted nodes rooted at the deleted entry, in which case these edits would be ignored. Additionally, an entry may not be deleted if it is updated, moved or is a primary copy when secondary copies exist as that would violate requirements 1, 4 or 5. If an entry with hangons is removed, the hangons should be moved to the preceding entry.

Listing 7 shows the pseudocode for a procedure that updates M_1 and M_2 and deals with the issues mentioned above. Step 11, which deals with checking for edits in the tree of deleted nodes is elaborated further below.

The delete tree T_D of a deleted node n contains n and all deleted descendants of n, without any non-deleted nodes in between. We define the *delete tree* of n to be the subtree T_n, from which the subtrees $T_{n_1} \ldots T_{n_k}$ are removed, where the nodes $n_i \in T_n$ are fully matched to a base node and have a partner, i.e. nodes that are guaranteed to be visited at another stage of the algorithm. The delete tree is illustrated in figure 7.4.

Now T_D is the set of nodes that will not appear in the merged tree, due to the removal of a node n from the merge list. We thus need to check that none of these nodes represent an edit (i.e. update, copy, move or insert), and if one (or more) does, we can either abort or issue a warning about updates possibly being lost. Checking the nodes for edits is done according to the definitions of edit operations in section 5.3.

Listing 7 Pseudocode for `removeDeletedOrMoved`

```
procedure removeDeletedOrMoved( M1, M2, b )
for each node b_i in b do
  if b_i has a match present in either M1 or M2 (but not both) then
    let n_i be the match of b_i and M := the list n_i is present in
    let e := the entry in M, whose node is n_i
    if n_i is moved or updated then
      abort due to conflict {delete/move or delete/update conflict}
    end if
    if n_i is a primary copy and secondary copies exist in M then
      abort due to conflict {delete/copy conflict}
    end if
    check the delete tree rooted at n_i for updates
    if e has hangons then
      append the hangons to the hangons of the entry preceding e in M
    end if
    if e is locked then
      lock the entry preceding e
    end if
    remove e from M
  end if
end for
endproc
```

Turning to our example (figure 7.3 on page 79) the result of running `removeDeletedOrMoved` on the merge lists (equations 7.3 and 7.4) is the following:

$$M_1^D = \underline{\alpha \quad b \quad c} \quad e \quad \underline{\overset{\overset{b}{i}}{f} \quad g} \quad h \quad \omega \tag{7.5}$$

$$M_2^D = \alpha \quad b' \quad c \quad \underline{e \quad h \quad f \quad \overset{b_1}{g} \quad \omega} \tag{7.6}$$

where M_1^D and M_2^D are the merge lists M_1 and M_2 after `removeDeletedOrMoved` has been run.

As can be seen d has been removed from M_1, since it was deleted in T_2 and a has been removed from M_2, due to the far move in T_1. Note that as a result of this, there are exactly the same entry nodes in both lists: b (updated to b' in T_2), c, e, h, f and g.

Combining the merge lists into a merge pair list

In this final step of the merge pair list generation, we combine the merge lists M_1^D and M_2^D into a merge pair list, obeying the sequencing of hangons and locked entries. We are now able to see the benefit of hangons, and the preceding delete stage: the entries in M_1^D and M_2^D now have a 1-to-1 correspondence, i.e. the node of each entry in M_1^D is partner to exactly one entry node in M_2^D and vice versa. This 1-to-1 correspondence considerably simplifies the merging of the lists.

We use a concurrent traversal of the merge lists, similar to the concurrent node traversal described earlier. Let p_1 and p_2 be the current position in M_1^D and M_2^D respectively, both initialized to the first position (i.e. the α entry) of the lists. We output the nodes p_1 and p_2 point to as well as their hangons as merge pairs . The position of p_1 and p_2 is then updated so that we always follow a right locking if one exists. This is repeated until the end of the merge lists is reached. The pseudocode in listing 8 covers the details of the procedure.

Listing 8 The `makeMergePairList` procedure

```
procedure makeMergePairList( M_1, M_2 )
let p_1 := 0, p_2 := 0
let M := an empty merge pair list
while not done do
  if M_1(p_1) and M_2(p_2) are right locked then
    if M_1(p_1 + 1) is not partner to M_2(p_2 + 1) then
      abort due to sequencing conflict
    else
      increment p_1 and p_2
    end if
  else if M_1(p_1) is right locked then
    increment p_1, p_2 := index of the partner of M_1(p_1) in M_2
  else if M_2(p_2) is right locked then
    increment p_2, p_1 := index of the partner of M_2(p_2) in M_1
  else
    increment p_1 and p_2 {Neither one was locked}
  end if
  if both M_1(p_1) and M_2(p_2) have hangons then
    abort due to sequencing conflict
  end if
  if M_1(p_1) = ω then
    done
  end if
  append {M_1(p_1), M_2(p_2)} to M
  if M_1(p_1) or M_2(p_2) have hangons then
    let e be the entry with hangons,and M'_1 its merge list
    let M'_2 be the other merge list
    for each hangon h of e do
      if h is a secondary copy then
        append {h, the partner in M'_2 of the primary copy of h} to M
      else if h is far moved then
        append {h, its structural partner in the other tree} to M
      else
        append {h, ·} to M
      end if
    end for
  end if
end while
return M
endproc
```

Resuming the example in figure 7.3 on page 79 we run `makeMergePairList` on the merge lists M_1^D and M_2^D (equations 7.5 and 7.6 on page 82) generated in the previous stage. The resulting merge pair list is:

$$\mathbf{M} = \begin{bmatrix} b & c & e & h & f & i & b & g & b_1 \\ b' & c & e & h & f & \cdot & b' & g & b_1 \end{bmatrix} \tag{7.7}$$

where the nodes in the top row are from T_1 and the nodes in the bottom row from T_2 (this is a matter of formatting the list, the algorithm adds pairs $\{n, m\}$ so that $n \in T_1$ and $m \in T_2$ as well as $n \in T_2$ and $m \in T_1$).

We note that the order of the pairs in the merge list satisfies the lockings in M_1^D and M_2^D, as well as leaves the unlocked nodes in their original order. Furthermore, the node i inserted in T_1 has no merge pair, and both copies of b in T_1 are paired up with the updated version of the node (i.e. b') in T_2.

Although it is clear the ordering of the pairs follows the ordering implied by the locked entries in the merge lists, some properties, such as termination of the loop, are not obvious. In section 7.2 we will prove that the generation of the merged pair list indeed behaves as it intuitively should.

Generating the merge pair list directly from the child list of a node n.

As mentioned before, we do not need to perform a merge of child lists at all if one of the cursors is not active, but can generate the merge pair list directly. Let C be the active cursor and n the node it points to. Generating the merge pair list is now a simple matter of iterating over $\mathbf{n}$ and filling in the structural partners of the nodes, as shown in listing 9.

Listing 9 Listing for `mergeListToPairList`

procedure mergeListToPairList(Node n)
let M := an empty merge pair list
for each node $n_i \in \mathbf{n}$ **do**
  if n_i is inserted **then**
    append(n_i, $\cdot$) to M
  else
    let n' := a full partner of n_i
    append $\{n, n'\}$ to the pair list
  end if
end for
return M
endproc

7.1.2 Additional conflict checking

There are some conflict situations not detected by the algorithm described above. Although the algorithm implicitly resolves these conflicts with its default behavior, we would typically still like to be notified of them. The input of the checking algorithm is the two merge lists (M_1, M_2) to be combined (before the call to `removeDeletedOrMoved`) and the common base node, b, of the parents of the nodes in the merge lists.

We do the following additional checks for each node $b_i \in \mathbf{b}$:

- If b_i is moved outside the merge list in either branch, and moved inside the merge list in the other branch, a move/move conflict has occurred (i.e. the node has been moved to two different locations)
- If b_i is moved outside the merge list in either branch, and deleted in the other, a move/delete conflict has occurred (i.e. the node was deleted and moved)
- If b_i is moved outside the merge list in both branches, we need to check that the destinations are the same. Otherwise a move/move conflict has occurred.

Match type (u_i, b_k)	Match type (v_j, b_k)	Content of w
u_i not present	Any	v_j
Any	v_j not present	u_i
Structural	Content	u_i
Structural	Structural	cmerge(u_i, v_j)
Content	Structural	v_j
Content	Content	cmerge(u_i, v_j)

Table 7.1: Content merging rules.

- If b_i is locked in one branch and deleted in the other, a delete conflict is issued.

7.1.3 Merging the contents of nodes u_i and v_j

The task at hand is to determine the node content that best represents the merged content of the nodes u_i and v_j, both, either or none of which may be updated with respect to the common base node b_k. Furthermore, u_i and v_j may be matched to b_k in several ways. We will start by analyzing in which cases we use the content of either node directly, and in which cases a content merge is needed. The merged node is denoted with w, and the function merging content by cmerge$(\cdot, \cdot)$.

1. u_i or v_j is not present.

   Let n be the node present (i.e. u_i or v_j). Since n has no merge pair, it has been inserted, and it's content should be used directly in w: content(w) = content(n)

2. $structuralMatch(u_i, b_k)$ and $contentMatch(v_j, b_k)$.

   Edits have been made to the content of u_i, that should only appear below u_i. Edits may also have been made to the content of v_j. The natural matching criterion guarantees that there is at least one structural match m for v_j in T_1 (unless v_j is deleted in T_1), and hence that any edits to of v_j will be honored when visiting the pair $\{v_j, m\}$ (or a warning issued if v_j is deleted). Thus, we should use the content of u_i directly, and furthermore, we may do so without the risk of ignoring edits. Thus, content(w) = content(u_i).

3. $structuralMatch(u_i, b_k)$ and $structuralMatch(b_j, b_k)$.

   Edits have been made to the contents of u_i, that should only appear below u_i. Edits have also been made to contents of v_j, that should only appear below v_j. Since the edits to the content of u_i and v_j should not be propagated, and are possibly unique for their respective node, we should merge the content: content(w) = cmerge(content(u_i), content(v_j)).

4. $contentMatch(u_i, b_k)$ and $structuralMatch(v_j, b_k)$.

   Using the same reasoning as in case 2) we conclude that content(w) = content(v_j).

5. $contentMatch(u_i, b_k)$ and $contentMatch(v_j, b_k)$.

   u_i and v_j are content partners, and we should to combine their contents. Thus, content(w) = cmerge(content(u_i), content(v_j)).

The rules for when to merge the content of u_i and v_j are summarized in table 7.1.

Operation in u_i	Operation in v_j	Insert
-	-	$(a, v) \in u_i$
-	Delete	-
-	Change	$(a, v) \in v_j$
Delete	-	-
Delete	Delete	-
Delete	Change	Conflict
Change	-	$(a, v) \in u_i$
Change	Delete	Conflict
Change	Change	Conflict

Table 7.2: Table of actions when merging attribute lists. A dash in the operations columns means no operation, and a dash in the insert column indicates that no (attribute,value) pair is appended to the attribute list.

The function cmerge$(\cdot, \cdot)$

Let u_i and v_j be the nodes whose content is to be merged and b_k their common base node. If both nodes are text nodes, the merged content is the content of the node changed with respect to b_k (if neither node is changed, the content can be set to either that of u_i or v_j). If both nodes have changed, a conflict has occurred.

If both nodes are element nodes we need to merge the element names as well as any attribute, value pairs present in the attribute list. We proceed in the following manner:

1. Let n be the node whose element name has changed with respect to b_k. If neither has changed we arbitrarily set n to u_i and if both have changed a conflict has occurred. The name part of the merged node is now the name part of n.

2. Let U be the set of attributes from u_i and B be the set from b_k. The attribute $a \in U$ is *inserted* in u_i iff $a \notin B$ and the attribute $a_b \in B$ *deleted* in u_i iff $a_b \notin U$. The attribute $a \in U$ is *changed* iff it is not inserted and if its value in u_i and b_k differs. Delete, insert and change with respect to v_j are defined in the same manner. The attribute list of the merged node is created thusly:

   (a) Add any inserted attribute, value pair (a, v) to the list. Duplicates are only added once. If there exists any inserted pairs (a_1, v_2) and (a_2, v_2) so that $a_1 = a_2$ but $v_1 \neq v_2$ a conflict has occurred.

   (b) Loop trough the pairs $(a, v) \in b_k$, inserting (attribute,value) pairs according to table 7.2.

   Informally, we want to merge the attribute lists so that attributes added in either node are also added to the merged attribute list (provided they do not conflict), any changed value should be used, and if the attribute was deleted from either node, it should not be in the merged attribute list.

7.1.4 Updating the cursors C_1 and C_2

When updating the cursors for T_1 and T_2 we must take into consideration that either node in the merge pair $\{u_i, v_j\}$ may not be present, and furthermore, the types of the matches (u_i, b_k) and

Match type of (u_i, b_k)	Match type of (v_j, b_k)	Assignments of C_1 and C_2
u_i is not present	Any	$C_1 = \emptyset$, $C_2 = v_j$
Any	v_j is not present	$C_1 = u_i$, $C_2 = \emptyset$
Content	Structural	$C_1 = u_i$, $C_2 = \emptyset$
Content	Content	$C_1 = u_i$, $C_2 = v_j$
Structural	Content	$C_1 = \emptyset$, $C_2 = v_j$
Structural	Structural	$C_1 = u_i$, $C_2 = v_j$

Table 7.3: Rules for updating the cursors C_1 and C_2 from the pair $\{u_i, v_j\}$.

(v_j, b_k). We will now go trough the different cases. Recall that b_k is the common base node of u_i and v_j.

1. u_i or v_j is not present.

   Let n be the node present (i.e. u_i or v_j). This means that n has been inserted in T_1, and thus there is no partner to merge it with. Assign $C_1 = n$ and $C_2 = \emptyset$.

2. $contentMatch(u_i, b_k)$ and $structuralMatch(v_j, b_k)$.

   Edits have been made to the child list of u_i, that should only appear below u_i. Edits may also have been made to the child list of v_j. The natural matching criterion guarantees that there is at least one structural match m for v_j in T_1 (unless v_j is deleted in T_1), and hence that any edits to the child list of v_j will be honored when visiting $\{m, v_j\}$ (or a warning issued if v_j is deleted). Thus, we should use the child list of u_i directly, and furthermore, we may do so without the risk of ignoring edits. Assign $C_1 = u_i$ and $C_2 = \emptyset$.

3. $contentMatch(u_i, b_k)$ and $contentMatch(v_j, b_k)$.

   Edits have been made to the child list of u_i, that should only appear below u_i. Edits have also been made to the child list of v_j, that should only appear below v_j. Since the edits to the child lists of u_i and v_j should not be propagated, and are possibly unique for their respective node, we need to try to merge the child lists. Thus, assign $C_1 = u_i$ and $C_2 = v_j$.

4. $structuralMatch(u_i, b_k)$ and $contentMatch(v_j, b_k)$.

   According to the same reasoning as in case 2) we should assign $C_1 = \emptyset$ and $C_2 = v_j$.

5. $structuralMatch(u_i, b_k)$ and $structuralMatch(v_j, b_k)$.

   This means u_i and v_j are structural partners, and we should try to merge their child lists: $C_1 = u_i$ and $C_2 = v_j$.

The rules for updating the cursors are summarized in table 7.3.

The reader is also encouraged to compare the rules for updating the cursors with those for merging content. The rules are similar if the meanings of $contentMatch(\cdot, \cdot)$ and $structuralMatch(\cdot, \cdot)$ are interchanged, as assigning both cursors a non-null node effectively means "merge child lists".

7.2 Analysis of the algorithm

In this section we will analyze some aspects of the algorithm in more detail. We begin by sketching a proof of correctness with respect to the requirements listed in section 5.3, focusing on the nontrivial steps. Following this, in section 7.2.2, we derive a worst-case time bound for the algorithm. Finally, we will examine the termination conditions of the algorithm.

7.2.1 Correctness of the algorithm

It is quite straightforward to see that the merge lists generated from the node child lists and processed by the `removeDeletedOrMoved` procedure contains exactly the node subsequences implied by requirements 3–5 on page 54 as combinations of locked and hangon nodes, and furthermore, that the non-locked entries are the same (and in the same order) as stated in requirement 6. It is also easy to see, by inspection of the pseudocode for `makeMergePairList`, that that if the start node of such as subsequence from either list appears in the merged pair list, the entire subsequence appears, as `makeMergePairList` always follows lockings, and expand the hangons after the node of the entry.

There are also some properties of the merge pair list generated by `makeMergePairList`, necessary for correct operation, that are not immediately obvious:

1. All nodes from the merge lists appear exactly once in the merge pair list
2. All sequencing conflicts are detected
3. The order of nodes not locked in either merge list is preserved.
4. The procedure always terminates

We will now prove these properties. We start by pairing up the entries from the merge lists M_1^D and M_2^D into merge list entry pairs $\{u_i, v_j\}$.

where u_i and v_j are the nodes of the entries $e_1 \in M_1^D$, $e_2 \in M_2^D$ and u_i and v_j are partners. As proved in the following lemma, each entry in M_1^D corresponds to exactly one entry in M_2^D, and vice versa, so that the nodes of the entries are partners. Thus, the entries of M_1^D and M_2^D pair up exactly.

Lemma 1 Each entry $e_1 \in M_1^D$ corresponds to exactly one entry in $e_2 \in M_2^D$ in the sense that the nodes of e_1 and e_2 are partners.

Proof We start by proving that $e_1 \in M_1^D$ corresponds to exactly one entry $e_2 \in M_2^D$. The proof is by contradiction.

First, we pair up the α and ω entries, of which there are one per list.

Assume there is no partner v_j for u_i in M_2^D. Then either u_i must be inserted or the base node b_k of u_i is not matched in M_2^D. The former case contradicts the construction of the merge list (the merge list does not have inserted nodes as entries) and the latter the result of the result of running `removeDeletedOrMoved` (u_i is far moved or deleted in T_2, in which case it is removed from M_1^D).

Assume there are several entries in M_2^D that are partners to u_i. This means the base node b_k that u_i is matched to has several matches, i.e. is copied, in M_2^D. This contradicts the construction of the merge lists, as only the primary copy is the node of an entry; all secondary copies are hangons.

Proving that $e_2 \in M_2^D$ corresponds to exactly one entry $e_1 \in M_1^D$ follows exactly the same reasoning.

Subseqence in M_1^D	Subseqence in M_2^D	Successors
$n_i n_k$	$m_i m_k$	$\{n_k, m_k\}$
$\underline{n_i n_k}$	$m_i m_l$	$\{n_k, m_k\}$
$n_i n_k$	$\underline{m_i m_l}$	$\{n_l, m_l\}$
$\underline{n_i n_k}$	$\underline{m_i m_k}$	$\{n_k, m_k\}$
$\underline{n_i n_k}$	$\underline{m_i m_l}$	$\{n_k, m_k\}$,$\{n_l, m_l\}$

Table 7.4: Table for determining the successor of the pair $\{n_i, m_i\}$ except $\{\omega, \omega\}$. In the table $k \neq l$.

Subseqence in M_1^D	Subseqence in M_2^D	Successors
$n_k n_i$	$m_k m_i$	$\{n_k, m_k\}$
$\underline{n_k n_i}$	$m_l m_i$	$\{n_k, m_k\}$
$n_k n_i$	$\underline{m_l m_i}$	$\{n_l, m_l\}$
$\underline{n_k n_i}$	$\underline{m_k m_i}$	$\{n_k, m_k\}$
$\underline{n_k n_i}$	$\underline{m_l m_i}$	$\{n_k, m_k\}$,$\{n_l, m_l\}$

Table 7.5: Table for determining the predecessors of the pair $\{n_i, m_i\}$ except $\{\alpha, \alpha\}$. In the table $k \neq l$.

We now have a set of entry pairs, which contains all the entry nodes from M_1^D and M_2^D. When visiting a pair, we output the nodes of the pair as well as any hangons. Thus, if all pairs are visited exactly once by `makeMergePairList` each node in the merge lists is appended exactly once to the merge pair list (property 1). We will now prove that the pairs are indeed visited exactly once, by proving that the pairs form an unambiguous list, in which each pair has exactly once successor (except the end pair $\{\omega, \omega\}$) and exactly one predecessor (except the start pair $\{\alpha, \alpha\}$).

We start by defining the predecessor and successor of an entry pair.

Pair successor Assume the entry pairs are ordered using some scheme, and the i:th pair is $\{n_i, m_i\}$ where $n_i \in M_1^D$ and $m_i \in M_2^D$. The successors of the pair $\{n_i, m_i\}$ is defined in table 7.4. Note that n_i and m_i always have succeeding entries in their merge lists, and hence there is always at least one successor. Also note that the case $n_i n_k$, $m_i m_l$ does not occur, as in that case either n_k or m_l is out of sequence with respect to **b**, and hence locked. The pair $\{\omega, \omega\}$ has no successor.

Pair predecessor Assume the entry pairs are ordered using some scheme, and the i:th pair is $\{n_i, m_i\}$ where $n_i \in M_1^D$ and $m_i \in M_2^D$. The predecessors of the pair $\{n_i, m_i\}$ is defined in table 7.5. Note that n_i and m_i always have preceding entries in their merge lists, and hence there is always at least one predecessor. Also note that the case $n_k n_i$, $m_l m_i$ does not occur, as in that case either n_i or m_i is out of sequence with respect to **b**, and hence locked. The pair $\{\alpha, \alpha\}$ has no predecessor.

According to the definitions, each pair has at least one predecessor (except $\{\alpha, \alpha\}$, which has none) and at least one successor (except $\{\omega, \omega\}$, which has none).

Lemma 1 If no entry pair has more than one successor nor more than one predecessor the pairs form an unambiguous list starting with $\{\alpha, \alpha\}$ and ending in $\{\omega, \omega\}$.

Proof By definition, each pair (except $\{\omega, \omega\}$) has at least one successor and (except $\{\alpha, \alpha\}$) one predecessor. According to the assumption of the lemma, no pair has more than one predecessor or successor. Thus, each pair (excluding $\{\omega, \omega\}$) has exactly one predecessor and (excluding $\{\alpha, \alpha\}$) one successor. The pair $\{\alpha, \alpha\}$ has one successor and $\{\omega, \omega\}$ has one predecessor. Thus, an unambiguous list starting with $\{\alpha, \alpha\}$ and ending in $\{\omega, \omega\}$ is formed.

Lemma 2 A sequencing conflict (as defined in section 5.3) is equivalent to an entry pair having more than one successor or predecessor.

Proof We first prove that "sequencing conflict $\Rightarrow$ more than one predecessor or successor", and the prove "more than one predecessor or successor $\Rightarrow$ sequencing conflict"

We prove "sequencing conflict $\Rightarrow$ more than one predecessor or successor" by proving the contrapositive, i.e. "not more than one predecessor and not more than one successor $\Rightarrow$ no sequencing conflict".

Since each pair has at least one successor or predecessor (except for the start and end nodes), we only need to examine the case with exactly one predecessor and successor. In this case we can, according to lemma 1, form an unambiguous list of entry pairs. Furthermore, since the definitions of pair successor and predecessor obeys the lockings in M_1^D and M_2^D, as well as the original node ordering when no locks exist, the linked list is a merged child list, as defined in section 5.3, and since such a list has been constructed, there is no sequencing conflict.

We continue by proving that "more than one predecessor or successor $\Rightarrow$ sequencing conflict"

A pair p has two successors p_1 and p_2 iff two subsequences $n_i n_k$ and $m_i m_l$, where n_i and m_l are not partners, are locked in the merge lists. Clearly, the subsequences $n_i n_k$ and $m_i m_l$ cannot both appear in the merged child list if there may only be one occurrence of the merged node of p (see the definition of merged child list), and thus it is impossible to construct a merged child list, leading to a sequencing conflict.

Using the same reasoning for a pair p that has two predecessors p_1 and p_2, we can show that it too leads to a sequencing conflict.

When checking for sequencing conflicts, we need in fact only check if a pair has more than one successor, as proved by the following lemma:

Lemma 3 If there exists a pair with 2 predecessors, there also exists a pair with two successors.

Proof Let p_1 and p_2 be the two predecessors. Since each pair has at least one predecessor, including p_1 and p_2 we can follow the predecessors of p_1 and p_2 until both predecessor chains reach the $\{\alpha, \alpha\}$ pair. Let the chains be $c_1 = p_1 p_1^1 p_1^2 \ldots \alpha$ and $c_2 = p_2 p_2^1 p_2^2 \ldots \alpha$. Assume the c_1 and c_2 have the last i, $i \geq 1$ pairs in common. Now the i:th last pair has 2 successors.

Since we only need to check for conflicting successors and hangons, we can do it at the same time as traversing the pair list.

Using lemmas 1-3 it is easy to prove the previously stated properties of the merge pair list:

Theorem 1 All nodes from the merge lists appear exactly once in the merge pair list

Proof When building the merge pair list we check that no pair has two successors. According to the contrapositive of lemma 3, this implies that no pair has two predecessors either. Thus the conditions of lemma 1 are satisfied, and the entries form an unambiguous list. By traversing this list, expanding the hangons from each entry, we have constructed a merge list in which each node from the merge pair lists occurs exactly once.

Theorem 2 All sequencing conflicts are detected.

Proof According to lemma 2, the conditions "more than one successor or more than one predecessor" and "sequencing conflict" are equivalent. Thus, a sequence conflict implies that there is a pair with more than one predecessor or successor, which by lemma 3 implies there's a pair with more than one successor, which is detected by `makeMergePairList`.

Theorem 3 The order of nodes not locked in either merge list is preserved.

Proof Follows from the definition of pair successor and predecessor.

Theorem 4 The procedure `makeMergePairList` always terminates

Proof The procedure starts with the pair α, calculating the successor pair on each iteration. If multiple successors exist the procedure aborts, otherwise the pairs form an unambiguous list (by lemma 1), which is finite. The procedure then terminates when reaching ω.

We have now ensured us of the correct operation of `makeMergePairList`, with respect to requirements 3–5. The child lists are only merged if their parents are structural partners, otherwise the child list pointed to by the (only) active cursor is used, fulfilling requirement 6 as well.

We now argue why deleted nodes do not appear in the merged tree (requirement 2). We observe that each node inserted into T_M originates from the merge pair list. The merged pair list is either formed by combining two child lists, in which case deleted nodes are removed in the second step of the generation the merge pair list (unless a conflict occurs), or by directly using the child list of a node, in which case the node has no structural partner. In the latter case, all child nodes are considered to be moved (as their parent has changed), and deleting them in the other branch is a move/delete conflict. Thus, all deletes that do not lead to conflicts are carried out.

Finally, we address requirement 1. From step 4 of `treeMerge` (page 76) we see that all nodes added to T_M that have content partners have their content merged.

As we have argued for the correct implementation of all the requirements, we can now state that the algorithm behaves correctly. No formal proof of the correctness was given, though.

7.2.2 Complexity analysis

We will now derive a worst-case time bound for the merging algorithm, excluding the optional conflict handling of detecting modifications to deleted nodes (more precisely step 11 of `removeDeletedOrMoved` and section 7.1.2). We start the analysis by making a few observations.

Assume $T_B = (R; a)$, $T_1 = (R; (a; i_1 \dots i_n))$ and $T_2 = (R; a_1 \dots a_n)$ where a and a_i are structurally matched and $i_1 \dots i_n$ are inserted nodes. $T_M = (R; (a_1; i_1 \dots i_n) \dots (a_n; i_1 \dots i_n))$ is now the merged tree, containing $(n+2)(n+1) = \Theta(n^2)$ nodes. Since each node in the merged tree must be visited once, we cannot hope to derive a better time bound than $O(n^2)$, where n is the number of nodes in T_1 and T_2.[2]

How about calculating the complexity with respect to the number of nodes in the merged tree? Assume $T_B = (R; a_1 \dots a_n)$, $T_1 = (R; a_1 \dots a_n)$ and $T_2 = (R)$. Now $T_M = (R)$ and thus contains 1

[2] In fact, we can construct cases with $O(n^3)$, $O(n^4)$, ... nodes.

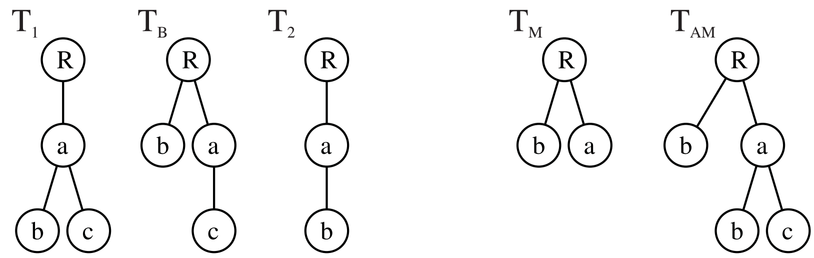


Figure 7.5: A merging example with the merged tree (T_M) and the augumented merge tree (T_{AM}).

node, regardless of n. Clearly, we cannot express the complexity with respect to the direct number of nodes in T_M either.

The main issue with using the nodes in T_M is that nodes deleted in the merged tree are not accounted for. As a help construct we define the *augmented merge tree*, which contains T_M and all deleted and, for reasons soon to be apparent, far-moved nodes at their original location, as well as their merged location. As complexity measure we will use the size of the augmented merge tree. Figure 7.5 illustrates a merge scenario with the merge tree T_M and the augumented merge tree.

We will denote the number of nodes in the augumented merge tree with N_M. Note that we do not modify the merging algorithm to generate the augumented merge tree; it is purely a theoretical aid in the analysis.

When making the analysis, we will make the same assumption we made earlier (assumption 2 in section 6.2.2) that there is an upper limit to the size of node content. This assumption will be used when determining the complexity of the content merge. We will also assume that each node in T_1 and T_2 has a list of pointers to its partners as well as to the base node (see section 6) allowing $O(1)$ lookup of matches and partners. Furthermore, we will make use of the set and map data structures defined in section 6.2.1.

Assume each call to `treeMerge` produces N_i nodes to the augumented merged tree, that the nodes u and v have n_u and n_v children, and finally that the common base node b of u and v has n_b children. If any of the aforementioned nodes does not exist, the corresponding node count is 0. The maximum of n_u, n_v and n_b is denoted with $\hat{n}$.

Lemma 1 A table generated in $O(n \log n) + O(n_b)$ can be used to determine if two nodes in $\mathbf{u}$ ($\mathbf{v}$) are in sequence in $O(1)$. n_b is the size of $\mathbf{b}$ and n the size of $\mathbf{u}$ ($\mathbf{v}$).

Proof To generate the help table, we add all children of u (v) to a set. We then loop over $\mathbf{b}$, adding backward pointers to the previously nondeleted node in $\mathbf{b}$ as we go along, checking for deletion by set lookups. Checking if two nodes u_i,u_j (v_i,v_j) are in sequence is now a matter of looking up the base match of u_i (v_i), following the backward pointer and see if that node is matched to u_j (v_j).

Adding all children to the set takes $O(n \log n)$. Looping over $\mathbf{b}$, adding backward pointers takes $O(n_b)$, for a total help table construction time of $O(n \log n) + O(n_b)$. Each step in the check takes $O(1)$ for a total of $O(1)$.

Lemma 2 The complexity of `makeMergeList` is $O(n \log n) + O(n_b)$, where n is the number of children of the node passed to the procedure

Proof Assume that the merge list is implemented as a linked list, in which appending can be done in time $O(1)$. Also assume that we keep a set of entry nodes added to the merge list, with insert and lookup taking $O(\log n)$.

Steps 1–4 take $O(1)$ time. The loop (step 5) is executed n times. The steps in the loop take the following times

- Step 6–11, 13–14, 18 take $O(1)$. Far moves and inserts can be detected by simple checks of the matchings.
- Step 12 takes $O(\log n)$ since we can see if the node is in the set of added entry nodes.
- Step 16 takes $O(\log n)$ since we need to add the node to the set of added entry nodes.
- Step 17 takes $O(1)$ if we use the help table described in lemma 1. The construction of the help table takes $O(n \log n) + O(n_b)$.

The total complexity of the loop becomes $n \cdot O(\log n) = O(n \log n)$. Steps 22–26 also take $O(1)$ time.

The total time of the procedure is thus $4 \cdot O(1) + O(n \log n) + 4 \cdot O(1) + O(n \log n) + O(n_b) = O(n \log n) + O(n_b)$, the two last terms due to the in sequence help table of step 17.

Lemma 3 The complexity of `removeDeletedOrMoved` is $O(\hat{n} \log \hat{n})$.

Proof Assume we have a two map data structures, one for M_1 and one for M_2. The maps have the base nodes of the entries in the merge lists as keys, and pointers to the entries as values. The maps can be constructed in $O(\hat{n} \log \hat{n})$. We will also need help tables for determining if a node is in sequence, as described in lemma 1. These are generated in $O(\hat{n} \log \hat{n})$ (notice that $O(n_b)$ is dropped since $n_b \leq \hat{n}$).

The loop (step 1) executes n_b times. The steps in the loop have the following complexities

- Step 2, 3 and 4 take $O(\log \max(n_u, n_v)) = O(\log \hat{n})$ using the aforementioned maps.
- Step 5 checks for movement or updates. Using hash values for equality comparison[3] and the in sequence help tables this can be accomplished in $O(1)$.
- Step 6, 9, 12, 15–18 take $O(1)$
- Step 11 is not included in this analysis (conflict handling)
- Step 8 takes $O(1)$ if primary copies are marked with a flag in `makeMergeList`.
- Step 13 takes $O(h_i)$, where h_i is the number of hangons added on this iteration of the loop.

Summing over loop yields

$$\begin{aligned} \sum_{i=1}^{n_b} 2 \cdot O(\log \max(n_u, n_v)) + 8 \cdot O(1) + O(h_i) &= \\ O(n_b \log \max(n_u, n_v)) + \sum_{i=1}^{n_b} O(h_i) &= \\ O(n_b \log \max(n_u, n_v)) + O(h) & \end{aligned} \tag{7.8}$$

[3] We assume each node has a hash value, which is calculated from its content.

where h is the total number of hangons and $h \leq \max(n_u, n_v)$. We can now write the complexity as

$$O(n_b \log \max(n_u, n_v)) + O(h) + 2 \cdot O(\hat{n} \log \hat{n}) = O(\hat{n} \log \hat{n}) \tag{7.9}$$

Lemma 4 $n_p \leq N_i$, where n_p is the number of nodes in either merge pair list.

Proof The nodes in the merge pair list is a subset of the nodes added to the augmented merge tree.

Lemma 5 $n_u \leq N_i$ and $n_v \leq N_i$.

Proof Each child of n_u or n_v is either present in the merge pair list, in which case it is also in the merged tree, or deleted or far moved, in which case it is present in the augmented merge tree.

Lemma 6 The complexity of `makeMergePairList` is $O(N_i \log N_i)$,

Proof As help structures we will use the same type of maps as in lemma 3 to look up the merge list entry from a node in the base child list. We use n_p to denote the number of nodes in the merge pair list.

Steps 1 and 2 are both $O(1)$.

The while loop, step 3, is executed maximally n_p times, as each iteration produces at least one merge pair. The steps inside the loop have the following complexities:

- Step 5,11,13: $O(\log \max(n_u, n_v))$ using the maps.
- The for loop of steps 27–35 has a complexity of $O(h_i \log \max(n_u, n_v))$. The primary copy can be found in $O(1)$ with a pointer from the hangon, initialized in `makeMergeList`. Step 29 takes $O(\log \max(n_u, n_v))$ using the maps. The other steps can also be implemented in $O(1)$.
- All other steps are $O(1)$.

Summing over the while-loop we get

$$\begin{aligned} \sum_{i=1}^{n_p} 3 \cdot O(\log \max(n_u, n_v)) + 15 \cdot O(1) + O(h_i \log \max(n_u, n_v)) &= \\ O(n_p \log \max(n_u, n_v)) + \sum_{i=1}^{n_b} O(h_i \log \max(n_u, n_v)) &= \\ O(n_p \log \max(n_u, n_v)) + O(h \log \max(n_u, n_v)) & \end{aligned} \tag{7.10}$$

where h is the total number of hangons and $h \leq \max(n_u, n_v)$. Thus, including the construction of the maps and using lemmas 4 and 5 the total complexity becomes:

$$\begin{aligned} O(n_p \log \max(n_u, n_v)) + O(h \log \max(n_u, n_v)) + & \\ O(\max(n_u, n_v) \log \max(n_u, n_v)) &\leq O(N_i \log N_i) \end{aligned} \tag{7.11}$$

Lemma 7 The complexity of `mergeListToPairList` is $O(\max(n_u, n_v))$

Proof Obvious from inspection of `mergeListToPairList`. The procedure simply loops over a merge list, expanding the hangons.

Lemma 8 $n_b \leq N_i$.

Proof Each child of b is present in both u and v, in which case it is in the merged tree, or not present in either u or v, in which case it is far moved or deleted, and thus present in the augmented merge tree.

Lemma 9 Generating the merge pair list has a complexity of $O(N_i \log N_i)$.

Proof If either of C_1 and C_2 are empty, the merge pair list is generated by `mergeListToPairList`, whose complexity is $O(\max(n_u, n_v) \log \max(n_u, n_v))$. Since $n_u \leq N_i$ and $n_u \leq N_i$ (lemma 5) $O(\max(n_u, n_v) \log \max(n_u, n_v)) \leq O(N_i \log N_i)$.

If C_1 and C_2 are nonempty, we call `makeMergeList` twice, followed by `removeDeletedOrMoved` and `makeMergePairList`. The combined complexity of this sequence is

$$\begin{aligned} O(n_u \log n_u) + O(n_v \log n_v) + 2 \cdot O(n_b) + & \\ O(\hat{n} \log \hat{n}) + O(N_i \log N_i) &= O(N_i \log N_i) \end{aligned} \tag{7.12}$$

using lemmas 2, 3, 6, 7 and 8.

Lemma 10 Merging the contents of two nodes has a complexity of $O(1)$.

Proof See section 7.1.3 for a description of content merging. As the size of the node content is bound by a constant (assumption 2) equality tests for strings can be made in $O(1)$, and the number of attributes is likewise limited to a constant (as storing an attribute requires more than 0 bytes of content). Thus, the problem is bounded by a maximum size and is thus $O(1)$.

Lemma 11 The for-loop in procedure `treeMerge`, with the exception of the recursive call, has a complexity of $O(n_p)$.

Proof The loop loops n_p times. Step 4 has a complexity of $O(1)$ according to lemma 10. Step 5 adds a node to the tail of the child list, which is also $O(1)$. Finally, step 6 consists of simple assignments, and hence is also $O(1)$. The complexity of the loop is now $n_p \cdot (O(1) + O(1) + O(1)) = O(n_p)$.

Lemma 12 The complexity of a call to `treeMerge` is $O(N_i \log N_i)$.

Proof The tree merge consists of generating the merge pair list and looping over it, the complexities of which are $O(N_i \log N_i)$ and $O(n_p)$, according to lemmas 9 and 11. Furthermore, since lemma 4 holds

$$O(N_i \log N_i) + O(n_p) = O(N_i \log N_i) \tag{7.13}$$

Theorem 1 The complexity of the merging algorithm is $O(N_M \log N_M)$.

Proof First we point out that each node in T_M is generated by exactly one of the calls to `treeMerge`. To see this, note that the algorithm only modifies T_M by adding nodes; it never modifies nodes in T_M. We can thus calculate the total complexity of the algorithm by summing over all the n calls to `treeMerge`.

$$\begin{aligned}\sum_{i=1}^{n} O(N_i \log N_i) &\leq \sum_{i=1}^{n} O(N_i \log N_M) \\ &\leq O(\log N_M) \sum_{i=1}^{n} O(N_i) \\ &\leq O(N_M \log N_M)\end{aligned} \tag{7.14}$$

noting that $\sum_{i=1}^{n} N_i = N_M$.

We now ask ourselves how the size of the augmented tree relates to the size of the real merged tree. We note that all nodes present in the augmented merge tree, but not present in the merge tree are either far-moved or deleted. Unfortunately, a deleted or far-moved node may appear several times in the augmented merge tree, due to the copy operation. To see this, consider the subtrees $(n;\, a\, b)$, $(n';\, a\, b\, c)$ and $(b;\, a\, b\, c)$, where n, n' and b are structural partners, and the subtree $(n';\, a\, b\, c)$ occurs several times. Each time the nodes n and n' are merged, the node c will be included in the augmented merged tree, but not in the merge tree, although it is only deleted once.

Let n_c be the maximum number of times `treeMerge` is called with the same pair of nodes $\{u, v\}$ (including empty pairs $\{u, \cdot\}$ and $\{\cdot, v\}$). $n_c - 1$ is now the number of extra copies (n_c is 1 if the node is not copied) of the merge of u and v that occurs in the merged tree. In the worst case, all far-moves and deletes occur in these copies, for a total of dn_c extra nodes in the augmented merge tree, where d is the total number of deletions and far-moves in T_1 and T_2 with respect to T_B. Thus,

$$N_M \leq n + dn_c \tag{7.15}$$

where n denotes the number of nodes in T_M. We can now write the complexity as

$$O((n + dn_c) \log(n + dn_c)) \tag{7.16}$$

7.2.3 Algorithm termination

Ideally, the algorithm would guarantee termination after a finite number of steps. Unfortunately, the algorithm does not have this property, as such: certain pathological matchings between the trees T_1, T_2 and T_B will cause the merged tree to be infinite, and hence the algorithm will not stop.

Consider the merge case depicted in figure 7.6. Starting from the root, we should merge the child lists a and d. The merged list is d. We then merge the child lists of d and its partner, which are b and the empty list. The merged child list becomes b. Next, the child lists of b and its partner are merged. The child lists are c and d, and the merged child list d.

We are now once again faced with the task of merging the child lists of d and its partner, leading to an infinite loop. The algorithm has “jumped back” to a previous stage of the merge. This jumping occurs when the nodes have been rearranged “vertically” (i.e. their depth has been rearranged). Compare this to rearranging the nodes “horizontally” in a child list, in which case the in-context locking causes a similar situation to generate a move/move conflict.

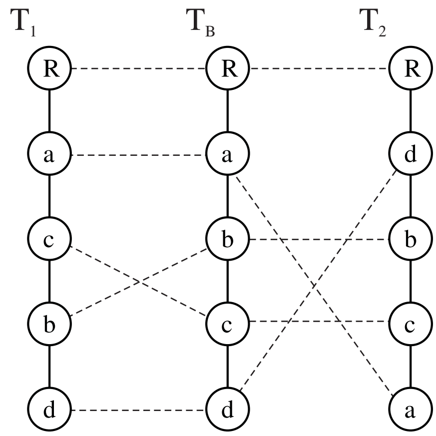


Figure 7.6: Matching leading to infinite merge loop (see merge case X6 in appendix G)

Possible remedies include storing the cursors passed as argument to `treeMerge` onto a call stack or developing a context-locking scheme similar to that used for nodes inside a child list. In the former case, we could check that the cursors passed to `treeMerge` are not already present on the call stack, in which case we have entered an infinite loop. To see this, note that the merged tree produced by `treeMerge` is fully determined by its arguments, and having the same argument pair occur twice on the call stack would thus imply that the merge tree being produced has itself as a subtree.

In this thesis, we have not developed a vertical locking scheme, nor implemented the the check for infinite looping, as the pathological matchings do not seem to arise in real world use. Instead a simple check for recursion depth has been used, which acts as a safeguard.

Chapter 8

The Tree Differencing Algorithm

The purpose of the tree diff algorithm is to encode the difference between two trees in a computer-friendly format, preferably as space-efficiently as possible. In tools based on edit scripts (such as [IAw01] and [CRGW96]) encoding is trivial: it is just a matter of storing the edit script. Our approach, which is not based on edit scripts, needs some more thought. In this chapter we will present our diff algorithm, which is based on encoding the matching between two trees.

8.1 Algorithm design considerations

We want the diff module to be reasonably fast and encode the differences in a manner that conserves space. We are not concerned about the diff making sense in terms of the differences a human user would perceive between the trees, as such user-friendly differencing can be accomplished by using the 3-way merge algorithm (see section 5.4.1).

We will not deal with issues of defining nor designing an optimal diff. An optimal diff is out of scope, as it would not address the research questions of this thesis and, furthermore, researching in this area would be enough work for a project of its own.

Existing differencing approaches that could be utilized includes the standard UNIX `diff` tool and methods of synthesizing an edit script from a tree matching (see e.g. [CG97b, CG97a]). The standard diff tool is not designed for structural data, leading to cases where a tree differencing tool can produced more compact diffs. For example, if subtrees are moved or copied between two versions, a tree differencing tool may be able to express these as "move" or "copy" operations, whereas `diff` would consider these to be deletions and insertions. Another issue is that `diff` may be tricked by changes in whitespace and line breaks that are not due to any real changes in the data.

Synthesizing an edit script from a matching is usually a straightforward albeit sometimes tedious process (see e.g. [Cha99, pp. 97-119]). The benefit of using a synthesized edit script is the possibility of generating, at the cost of increased complexity, more readable and more optimized diffs, neither of which we are interested in currently. The natural choice in this case is therefore to avoid any intermediate difference representations and simply encode the matching directly, in such a way that the target tree can be restored with knowledge of the encoded matching and the base tree.

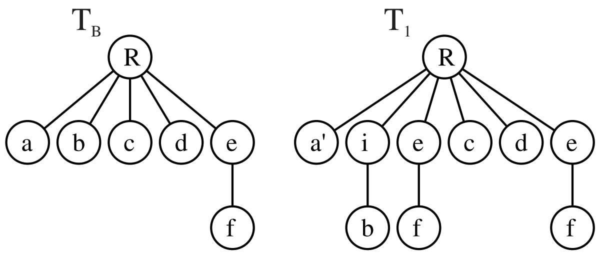


Figure 8.1: Two trees, T_B and T_1, whose difference we want to encode.

8.2 The algorithm

Assume that T_B and T_1 are the trees whose difference the algorithm should generate. The algorithm works by searching for subtrees in T_1 that are also present in T_B and encoding T_1 using references to these subtrees. Finding the subtrees in T_1 that are present in T_B is essentially the same as the the tree matching problem, presented in section 6, with only a few details differing. In this section we will also demonstrate how to patch T_B with the diff, i.e. generate T_1 from T_B and the diff.

The matching encoding that we have chosen imposes some limitations on the matching. We therefore start by introducing the method for encoding matchings, and then proceed by showing how to construct matchings that can be encoded using the method.

8.2.1 Encoding a matching as a diff

Assume that the nodes in T_B have been sequentially numbered according to the breadth-first traversal of T_B with the root having number 0 (for an example of BFS numbering, see figure 1.4 on page 5). Furthermore, assume we have a set of matched truncated subtrees between T_B and T_1, and where in each matched subtree the node matches are exact, i.e. a pair of nodes is matched on if the content of the nodes is equal. We can now think of T_1 as a tree consisting of subtrees copied from T_B and nodes not present in T_B, as illustrated in figure 6.6 on page 70.

Encoding of T_1 is now accomplished by writing out T_1 as an XML file where each matched subtree has been "collapsed" into a special `<copy>` tag, which references nodes in T_B. The main issue is how to express the subtrees copied from T_B, so that no structural information is lost.

Note that each truncated matched subtree is uniquely identified by its root and leaf nodes. For each copied subtree, it is thus sufficient to encode the root and the leaf nodes. Since all nodes in T_B have a unique number generated by the breadth-first numbering, we can reference nodes using their BFS number in T_B. A simple approach to encoding a copied subtree would be to use a `<copy>` tag for the root of the subtree, and a `<leaf>` tag for each leaf node. The children of each leaf node would be added below their corresponding leaf tags, as shown in figure 8.2.

We refine the space required for this simple encoding by noting that the children of a copied subtree are often a copied subtrees themselves, due to the way matched subtrees are constructed. We allow the `<copy>` tag to include a leaf node from the enclosing subtree copy operation, thereby omitting the need for a `<leaf>` tag if the child is the root of another subtree.

If the child is a node not present in T_B we will use an `<insert>` tag that indicates the destination of the node, by identifying the leaf node of the enclosing copy operation. The inserted node (or an entire inserted subtree) will be enclosed in the `<insert>` tag. The inserted subtree may also include `<copy>` tags, referencing subtrees from T_B. These `<copy>` tags may in turn include new `<insert>` tags etc.

```
<copy root="1">
  <leaf id="6">
     <child_of_leaf_node />
     ⋮
     <child_of_leaf_node />
  </leaf>
  <leaf id="7">
     <child_of_leaf_node />
     ⋮
     <child_of_leaf_node />
  </leaf>
</copy>
```

Figure 8.2: A trivial diff encoding.

```
<diff>
 <insert dst="1">
  <aP />
 </insert>
 <insert dst="1">
  <i>
   <copy src="3" />
  </i>
 </insert>
 <copy src="6" dst="1" />
 <copy src="4" dst="1" />
 <copy src="5" dst="1" />
 <copy src="6" dst="1" />
</diff>
```

Figure 8.3: Unoptimized diff of T_B and T_1 in figure 8.1.

The leaf node is encoded in the `dst` attribute of both tags, and the root of the subtree to copy from T_B in the `src` attribute of the copy tag. The children of a leaf node are listed in sequence in a single block. There is no restriction in which order the child lists of the leaf nodes appear.

In addition, since the roots of T_1 and T_B always match, the difference is assumed to implicitly start with a copy operation starting at the root of T_B. This has the advantage of producing an empty diff if T_1 and T_B are identical. Using the above mentioned encoding scheme the diff between the trees T_B and T_1 in figure 8.1 would be the one shown in listing 100.

We make one final observation to optimize the size of the diff. Consider the example in figure 8.1, where a few child nodes of the root node have been edited. We see that the other children are roots of matched (single node) subtrees, and furthermore, that these subtree roots occur as adjacent children in T_B. Since we have numbered the nodes in T_B using the BFS scheme the number of adjacent children are in sequence, leading to patterns of the type

```
<copy dst=m src=n />
<copy dst=m src=n+1 />
⋮
<copy dst=m src=n+k />
```

```
<diff>
 <insert dst="1">
  <aP />
 </insert>
 <insert dst="1">
  <i>
   <copy src="3" />
  </i>
 </insert>
 <copy src="6" dst="1" />
 <copy src="4" dst="1" run="3" />
</diff>
```

Figure 8.4: Optimized diff of of T_B and T_1 in figure 8.1

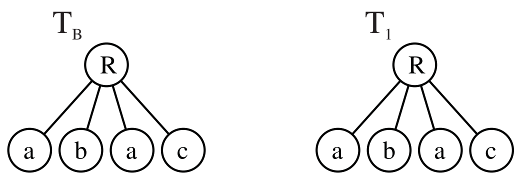


Figure 8.5: Trees that can be matched in several ways

in the output. We will encode these patterns as:

```
<copy dst=m src=n run=k+1 />
```

The optimized diff encoding of the above example is shown in listing 8.4.

Since the tree T_1 may contain arbitrary XML, it is also possible that it contains the special `<copy>` and `<insert>` tags. In such cases, the tags need to be escaped, lest they be interpreted as special tags. Escaping is accomplished by enclosing the offending tag inside an `<esc>` tag.

The pseudocode for generating the diff is listed in listing 10. The algorithm assumes that a matching of subtrees exist.

8.2.2 Generating the matching between T_1 and T_B.

The tree matching is constructed in a manner similar to that presented in section 6, with the exceptions that we only allow matches between nodes if the content matches exactly and that, in case of ambiguities, we select the match that does not break the sequence of source nodes, i.e. the sequence of `src` attributes in the `<copy>` tags. We only allow exact matches since the algorithm does not encode node content differences. For instance, consider that two nodes, $T_B(a)$ and $T_1(a)$, whose contents are "The meaning of life is 41" and "The meaning of life is 42". If these were matched, we would have to store the entire content of a, resulting in no space savings from matching similar nodes.

As for not breaking the sequence when solving ambiguities, consider the example depicted in figure 8.5. Using exact node matching we could match the a nodes in T_1 to any (including the same) a node in T_B; however, from the difference encoding point of view it makes sense to match the first a in T_1 to the first a in T_B and the second to the second, as we then have a sequence of copies with increasing source node number, which will be encoded as:

Listing 10 Pseudocode for generating tree diffs.

```
procedure diff( $T_B$, $T_1$ : Tree )
  BFS enumerate nodes in $T_B$
  copy( $T_B$, $T_1(R)$ )
endproc

procedure copy( $T_B$ : Tree; $n_1$ : Node )
  $T$ := matched subtree in $T_B$ to subtree rooted at $n_1$and $\mathbf{l}$ := leaf nodes in $T_1$ of $T$
  for each node $n$ in $\mathbf{l}$ do
    for each child $n_i$ of $n$ do
      if $n_i$ has a match $n_B$ in $T_B$ then
        output <copy src=$n_B$ dst=$n$>
        copy( $T_B$, $n_i$ )
      else
        output <insert dst=$n$>
        insert( $T_B$, $n_i$ )
      end if
    end for
  end for
endproc

procedure insert( $T_B$ : Tree; $n_1$ : Node )
  if $n_1$ needs escaping then
    output <esc>
  end if
  output $n_1$
  for each child $n_i$ of $n_1$ do
    if $n_i$ has a match $n_B$ in $T_B$ then
      output <copy src=$n_B$>
      copy( $T_B$, $n_i$ )
    else
      insert( $T_B$, $n_i$ )
    end if
  end for
endproc
```

```
<copy dst="0" src="1" run="4" />
```

Compare this to the case where the a nodes are matched “crosswise” (i.e. the first a matching the second and vice versa):

```
<copy dst="0" src="3" />
<copy dst="0" src="2" />
<copy dst="0" src="1" />
<copy dst="0" src="4" />
```

In practice ambiguities are solved by looking at the left sibling (if one is present) of the node being matched and the node it is matched to, call it m. If any of the candidate matches is a right sibling to m, then it is used.

8.2.3 Patching: Regenerating T_1 from T_B and the diff

Regenerating T_1 from T_B and the diff is straightforward: we recursively traverse the diff tree, expanding any `<copy>` tags encountered, using the `src` attribute and the leaf nodes encoded in the children of the tag. The process starts by initiating a copy operation of the subtree rooted at $T_B(R)$, in accordance with the implicit copy operation assumed in section 8.2.1. The algorithm is in listing 11.

Listing 11 The patching algorithm

```
procedure patch( T_B, D : Trees )
  create lookup table for nodes in T_B using their BFS numbering
  p := new node
  copy( p, D(R), T_B(R) )
  return p
endproc

procedure insert( p, d : Node )
  if d is not <esc>, <copy>, <insert> then
    n := new node, whose content is that of d
    add n as the last child of p
    for each child n_i of d do
      insert( n, n_i )
    end for
  else if d is <esc> or <insert> then
    insert( p, child of d )
  else if d is <copy> then
    s := node indexed by src attribute of d
    copy( p, d, s)
  end if
endproc

procedure copy( p, d, s : Node )
  L := leaf nodes gathered from dst attribute of d's children
  copy subtree from s to p stopping at leaf nodes L
  C := copies of the leaf nodes in the patched tree.
  for each child n_i of d do
    i := the node in C corresponding to the dst attribute of n_i
    insert( i, n_i )
  end for
endproc
```

8.3 Algorithm complexity

Worst-case performance of the differencing process is limited by the subtree matching, which is $O(n^2)$ (see section 6). Assuming the matching has been calculated the encoding and patch algorithms use linear time.

We first justify the claim that `procedure diff` has a complexity of $O(n)$, where n is the number of nodes in T_1 and T_B combined.

The differencing algorithm is basically a depth-first traversal of T_1 with the addition of collapsing copied subtrees. Depth-first traversal of a tree has linear complexity since each node in T_1 is visited exactly once in $O(1)$ time. The detection of copied subtrees is linear in the number of nodes in the subtree (see section 6.2.2), and each node in T_1 belongs to maximally one such subtree. Each node in T_1 produces a limited amount of output; in the worst case three nodes: `<insert>`, `<esc>` and the node itself. Numbering the nodes using breadth-first traversal is also $O(n)$.

The patching algorithm also has a complexity of $O(n)$, n being the number of nodes in $T_1 + T_2$. The patching algorithm performs a depth-first traversal of the diff tree, outputting either inserted nodes or a copied subtree, which each node in T_1 being encoded using a maximum of tree tags. Furthermore, each of the outputted nodes is present in T_1 and each node from T_1 is outputted exactly once.

Chapter 9

The Implementation of 3DM

In this chapter we present the prototype implementation of 3DM. In the first two sections we present the conflict and merge logging facilities which allow us to track the operation of the merging algorithm. Invocation of the program is presented in section 9.3, and a brief overview of the implementation is given in section 9.4. Section 9.5 presents XML-specific issues that are currently unimplemented.

9.1 Expressing and resolving conflicts

Conflicts during merge are detected according to the definition of the merge algorithm (refer to sections 7.1.1 and 7.1.3). Instead of aborting the merge, the 3DM implementation tries to resolve the conflict situation so that the algorithm can continue. Some conflict situations are not severe, and can successfully be recovered from, whereas others most likely will result in an unusable merged tree. The conflicts have therefore been divided into two categories: conflicts and conflict warnings.

The different types of conflicts and conflict warnings recognized by 3DM and how they are resolved are listed in tables 9.1 and 9.2.

For each conflict or conflict warning that occurs, 3DM logs a conflict message, the location of conflict in the merged tree as well as the location of the conflicting nodes in the base and branch trees (when available). The locations in the trees are expressed using the path syntax presented in section 1.1.3.

Conflicts are dumped to the file `conflicts.log` in XML format, which has the following structure:

- The root element is `<conflictlist>`
- The conflict list contains a list of conflicts (if conflicts have occurred) inside a `<conflicts>` element, and a list of warnings (if there are any warnings) inside a `<warnings>` element.
- The conflicts (and conflict warnings) are of type update, delete, insert and move, indicated by tags with the same names.
- Each conflict contains a text node describing the conflict, as well as 1–4 `<node>` tags describing the locations in the base, merged and branch trees. The `tree` attribute of the node tag indicates the tree (base, merged, branch1 or barnch2) and the `path` attribute the location in the tree.

Conflict Type	Description and Resolution	Reference
Update	**Description** A node was updated in both branches and the updates were not equal. **Resolution** Use the update from branch 1.	Table 7.2 on page 86.
Move (sequencing)	**Description** Conflicting moves inside a child list. **Resolution** Use the child sequencing of branch 1.	Listing 8 on page 83, lines 6 and 18
Move	**Description** Conflicting moves, one of the moves is far. **Resolution** Ignore the move inside childlist.	Section 7.1.2
Move	**Description** Far moves of a node in both barnches. **Resolution** Accept all moves, which implies thatextra copies will appear.	Section 7.1.2
Move	**Description** A node is far moved in one branch and deleted in the other. **Resolution** Ignore the delete.	Section 7.1.2
Delete	**Description** A node locked in context is deleted. **Resolution** Delete the node anyway.	Listing 7 on page 82, lines 5–10

Table 9.1: Table of conflicts recognized by 3DM and how they are resolved.

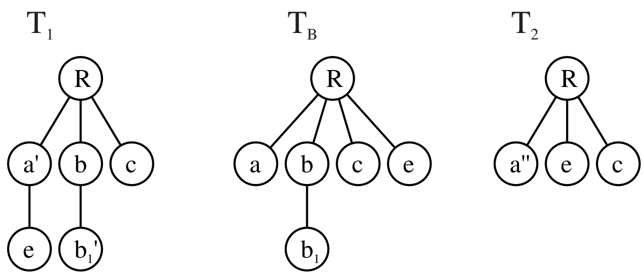


Figure 9.1: A merge situation leading to conflicts.

A sample conflict log from the merge of the trees depicted in figure 9.1 can be seen in listing 9.2. The conflicts that have occurred are:

1. The element name of the `<a>` node has been updated differently in the branches. Table 9.1 row 1, the `<update>` conflict in the log.
2. The `<e>` node has been moved to different locations. Table 9.1 row 3, the `<move>` conflict in the log.
3. The child of the `<b>` node was updated in one branch, and the subtree rooted at `<b>` deleted in the other. The update of the child node is thus lost. Table 9.2 row 4, the `<delete>` conflict warning in the log.

9.2 Logging merge operations

The merge logging facility, described section 5.4.1, captures the edits on the base tree performed by the merging algorithm as well as indicates the origin of these, i.e. which branch and what location

Warning Type	Description and Resolution	Reference
Update	**Description** A node was updated in both branches and the updates were equal. **Resolution** Use the update from branch 1	Table 7.2 on page 86.
Insert	**Description** Nodes were inserted/copied to the same location in both branches and the insertions/copies are equal. **Resolution** Use the insertions/copies from branch 1	Listing 8 on page 83, line 18
Insert	**Description** Nodes were inserted/copied in both branches and the insertions/copies are not equal. **Resolution** Insert the nodes from branch 1, followed by the nodes from branch 2.	Listing 8 on page 83, line 18
Delete	**Description** Edits in deleted subtree. **Resolution** Delete subtree anyway.	Listing 7 on page 82, line 11

Table 9.2: Table of conflict warnings recognized by 3DM and how they are resolved.

```
<?xml version="1.0" encoding="UTF-8"?>
<conflictlist>
 <conflicts>
  <move>
   Node moved to different locations-trying to recover by ignoring move
   inside childlist (copies and inserts immediately following the node
   may have been deleted)
   <node tree="merged" path="/0" />
   <node tree="base" path="/0/3" />
   <node tree="branch1" path="/0/0/0" />
   <node tree="branch2" path="/0/1" />
  </move>
  <update>
   Node updated in both branches, using branch 1
   <node tree="merged" path="/0/0" />
   <node tree="base" path="/0/0" />
   <node tree="branch1" path="/0/0" />
   <node tree="branch2" path="/0/0" />
  </update>
 </conflicts>
 <warnings>
  <delete>
    Modifications in deleted subtree.
   <node tree="merged" path="/0" />
   <node tree="base" path="/0/1" />
   <node tree="branch1" path="/0/1" />
  </delete>
 </warnings>
</conflictlist>
```

Figure 9.2: Conflict log from merging the trees in figure 9.1.

an edit originates from. The edits are expressed using the familiar insert, delete, update, move and copy operations.

The merge log contains entries for each of the nodes in the merged tree that have been edited in some way and entries for the deleted nodes in the base tree. Each entry is of the format `<`*operation* `path=`p `src=`b `originTree=`T `originList=`l `originNode=`n`>` where *operation* is the name of the operation, p is the location of the node in T_M, b is the location of source node in T_B, T is the tree from which the operation originates (T_1 if the operation occurs in both trees), l is the location of the node in T whose child list originated the operation (move, copy and delete operations) and n is the location of the node in T which originated the operation (used with update and insert operations). All locations are given using the aforementioned path syntax. The semantics of the entries are as follows:

- **Insert**

  For all inserted (as defined in section 5.3) nodes in T_M there is an entry in the edit log for which *operation* = insert and n is the location of the node (in either branch, as indicated by the `originTree` attribute) that was inserted. The `src` and `originList` attributes are not present.

- **Delete**

  For all delete trees, i.e. subtrees of deleted nodes (a deleted node is defined in 5.3) in T_B, there is an entry in the edit log for which *operation* = delete, b is the location of the root of the subtree and l is the location of the child list originating the delete. The `path` and `originNode` attributes are not present. A single delete entry thus signals the deletion of an entire subtree. The leaf nodes of the subtree are the parents of nodes mentioned in the `src` attribute of other edit operations, i.e. if a node inside a subtree mentioned by a delete entry is mentioned by any other edit entry, the subtree rooted at that node is not deleted.

- **Update**

  For all updated (as defined in section 5.3) nodes in T_M there is an entry in the edit log for which *operation* = update, n is the location of the node that originated the update, and b is the location of the updated node. The `originList` attribute is not present.

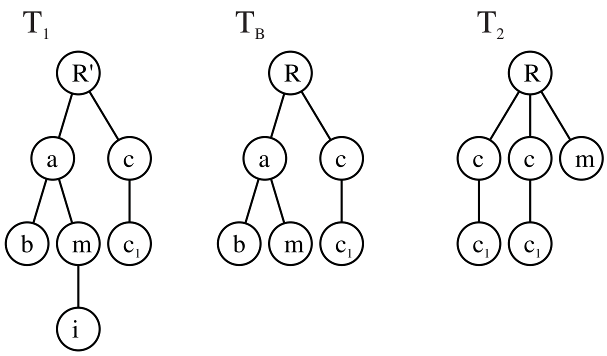
- **Move**

  For all moved (as defined in section 5.3) nodes in T_M there is an entry in the edit log for which *operation* = move, b is the root of the subtree being moved and n is the location of the node originating the move. Note that the move operation denotes movement of an entire subtree, not just a single node!

- **Copy**

  For all copied (as defined in section 5.3) nodes in T_M there is an entry in the edit log for which *operation* = copy, b is the root of the subtree being copied and n is the location of the node originating the copy. As with moves, the copy operation denotes copying of an entire subtree.

Figure 9.3 shows an example of merge logging. Notice that there is only one copy operation (as the leftmost subtree with root c is not moved), and how the delete only affects the nodes a and b, but not m, as it is addressed in the `<move ...>` entry of the edit log. Figure 9.4 is a screen shot




```
<?xml version="1.0" encoding="UTF-8"?>
<edits>
 <update path="/0" src="/0" originTree="branch1" originNode="/0" />
 <delete src="/0/0" originTree="branch2" originList="/0" />
 <copy path="/0/1" src="/0/1" originTree="branch2" originNode="/0/1" />
 <move path="/0/2" src="/0/0/1" originTree="branch2" originNode="/0/2" />
 <insert path="/0/2/0" originTree="branch1" originNode="/0/0/1/0" />
</edits>
```

Figure 9.3: Tree merging and merge log

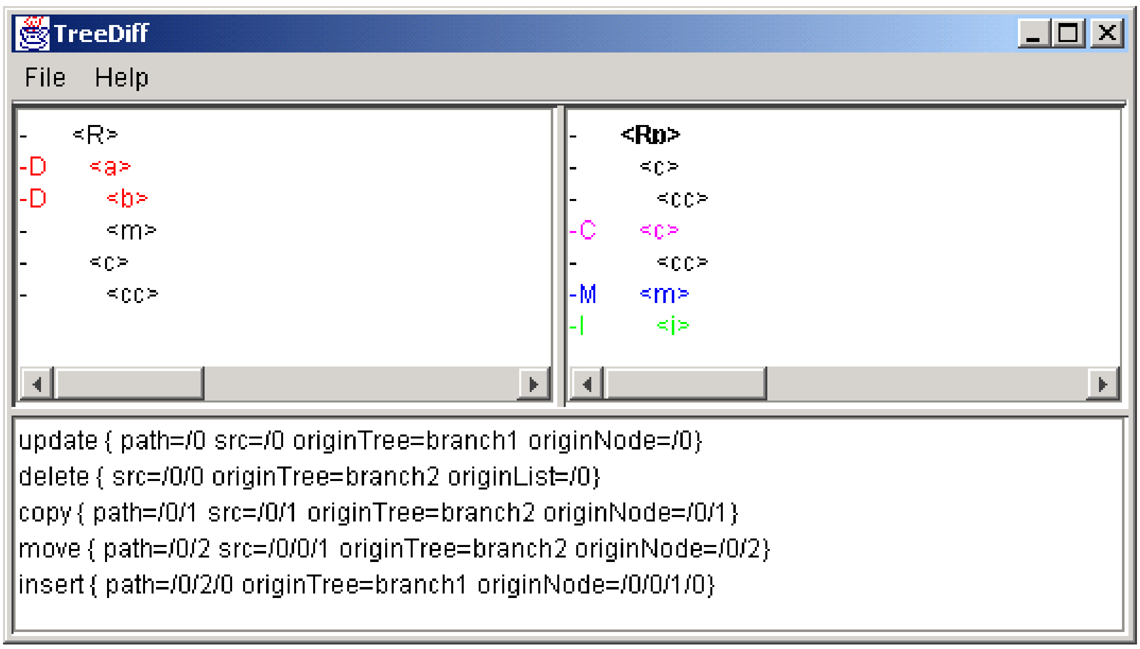


Figure 9.4: Screen shot from the TreeDiff utility that visualizes the edit log in figure 9.3. The left pane shows the original file (T_B) and the right the new version (T_M). The characters at the start of each line indicate the operation: Insert, Delete, Move and Copy. Updated lines are shown in bold font.

from a simple utility that visualizes the edits.

The edit logging (except deletes) is implemented in conjunction with the `makeMergePairList` procedure, where it is easy to check each node for edit operations. Delete operations are logged in `removeDeletedOrMoved`.

9.3 Invoking 3DM

3DM can be invoked in three modes: merging, differencing and patching. Using a wrapper script that calls Java with the appropriate classpath and main class the command-line syntax for 3DM is:

```
3dm [-e [logfile]] [-c threshold] {-m base branch1 branch2 |
                      -d base branch1 | -p base patch} [outfile]
```

Use the `-m` or `--merge` option to perform a 3-way merge of XML files, the `-d` (`--diff`) option to produce the difference between two XML files and the `-p` (`--patch`) option to patch an XML file. If no outfile is given the output is directed to standard out, otherwise to the output file given. To aid output piping, all messages are directed to standard error. The `-e` (`--editlog`) option causes an edit log to be generated when merging (default log file is `edit.log`). With the `-c` (`--copythreshold`) option you can specify the minimum information size of a copied subtree, default value is 128 bytes (see section 6.1.3).

9.4 Overview of the Java implementation

The prototype of 3DM was implemented in Java (rev 1.3 [Sun]). Java was selected due to its wide acceptance, clean object oriented model and well designed standard class library. Due to performance issues a production version of the tool should, however, preferably be implemented in some natively compiled language such as C/C++. The full Java source code for 3DM is released under the GPL and can be found in appendix H. In this section, we give a brief overview of the structure of the implementation. Those readers interested in the details of the implementation are advised to start by reading this section to get an overview and thereafter to read trough the source code.

The main class is `TreeDiffMerge`, which parses the command line, builds the tree matchings and runs the merge, diff or patch algorithms. Matching of trees for the purpose of merging is implemented in the `Matching` class, where you will find implementations of the `matchSubtrees`, `matchSimilarUnmatched`, `removeSmallCopies` and `setMatchTypes` procedures presented in section 6.1. Matching for differencing purposes (as defined in section 8.2.2) is implemented in the `DiffMatching` class. The `TriMatching` class encapsulates the two matchings between a common base tree and two branches.

The merging algorithm can be found in the `Merge` class. The implementation of the procedures mentioned in section 7.1 bear the same name as in the pseudocode (`treeMerge`, `makeMergeList` etc.). The rules for when to merge content (presented in section 7.1) are implemented in the `mergeNodeContent` method, content merging itself is implemented in the `cmerge` method. Cursor updating (section 7.1.4) is implemented in the `getRecursionPartners` method.

The differencing algorithm, as described in section 8.2.1, is implemented in the Diff class and the patching algorithm (section 8.2.3) is implemented in the Patch class. As in the other classes, the actual method implementations have the same names as the pseudocode procedures.

Conflict and edit logging are implemented in the ConflictLog and EditLog classes, with appropriate hooks in the merge algorithm for generating entries to the logs.

The XML-specific code is limited to the XMLParser and XMLPrinter classes. The former parses XML to the internal tree format (implemented in the Node, BaseNode and BranchNode classes) and the latter encodes internal tree structures back to XML (see figure 5.11 on page 56). The XMLParser class is a wrapper around any standard SAX [Meg00] parser; 3DM was tested with the Xerces 1.2 parser from the Apache XML Project [Apa01]. If 3DM is adapted for other structured file formats (e.g. HTML) these classes need to be replaced. Internally node content is stored in XMLNode objects which, despite the class name, can be used to store any tree structure adhering to the nodes-with-attributes paradigm.

9.5 Unimplemented XML features

Being a prototype, the current implementation of 3DM has some limitations in its applicability to any general XML file. The current parser module essentially uses the document tree as it was returned by the XML SAX parser (in this case Xerces), with only some minor adjustments (such as normalizing whitespace in text nodes). A full-featured XML version of the tool needs to perform some additional processing in order to work correctly on any XML document. Unimplemented issues include:

- **Document Type Definition (DTD) processing**

  The current implementation does not process the DTD, or even check that the DTDs of the documents processed are identical. Ideally the DTDs would be parsed as trees of their own, on which we may run the merge and diff algorithms.

  No DTD is output in the diff and merged files.

- **Processing of entities**

  The default parser expands XML entity references (such as < or perhaps &author; if such an entity was declared in the DTD) in elements, attributes and text nodes. A better approach would be to let the entities remain unexpanded and instead compare their declarations in the DTD.

- **Whitespace and CDATA processing**

  Whitespace is not considered significant in text, although it should in some cases, e.g. the <pre> tag in XHTML. If the XML document is validated, the xml:space attribute can be used to distinguish these cases. The marking of CDATA sections (<![CDATA[...) is not preserved in the output.

- **Comments**

  Comments are not processed, and stripped from the output. Comment processing could be added by introducing a special comment tag in the internal document tree.

It should be pointed out that none of these issues are due to limitations in the algorithms presented in this thesis, but due to the limited scope of the implemented prototype. The above limitation presented very few problems when running the XHTML, SVG and OpenOffice XML files included in the use cases (e.g. having to manually paste the DTD to the merged output).

Chapter 10

3DM Evaluation

In this chapter we will run 3DM on the merge and use cases and compare the results to those obtained by performing the merge by hand. We start with the merge cases and move on to the use cases. For each use case we will, in addition to analyzing the results, present the actual sequence of 3DM invocations in order to show exactly what steps were performed. Conclusions of the evaluation are presented in the final section of the chapter.

10.1 Merge cases

Out of 42 merge cases 40 were successful, taking into consideration that several of the merge cases permitted several reasonable outcomes of the merge. In the test runs the output produced by 3DM was accepted if it was one of the reasonable outputs of the case. For instance, when appending two items to a list, each branch containing an item, it is reasonable to accept items appended in either order. To check the symmetry of the merging algorithm, each of the cases were run with the branch file names both ways, i.e. `3dm -m b 1 2` and `3dm -m b 2 1`.

For the conflicting cases, the output was considered correct if the conflict was identified correctly.

Of the 3 failed cases, two failed because the edits in the child lists caused the node context requirements (see section 5.1.2) to be violated and the remaining one failed because the matching did not properly reflect the actual editing operations. We will now look at the failed cases in some detail.

Merge case A3 (figure 10.1) fails because all instances of the copied node s_1 are locked, as stated in section 5.1.2. We could relax the locking condition to only lock those copies that have actually been moved, in which case there would have been no problem. Still, we feel that a stricter locking policy is less error prone, as the first node in the list is not always necessarily the original instance.

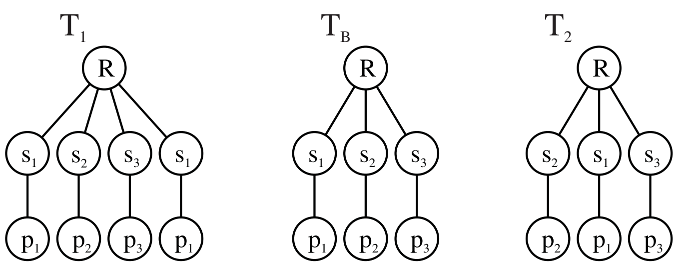


Figure 10.1: Merge case A3

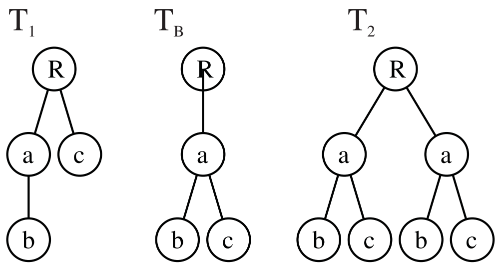


Figure 10.2: Merge case A8

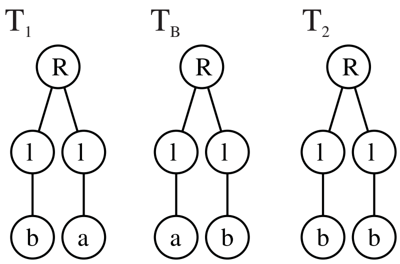


Figure 10.3: Merge case A10

In merge case A8 (figure 10.2) the problem is that copying the subtree rooted at a in branch 2, causes the last entries in the merge list to be locked (according to merge requirement 4):

$$\begin{array}{ccc} & a & \\ \alpha & \underline{a \quad \omega} & \end{array}$$

Thus, it is not possible to move c to then end of the list without violating the locking constraints. The case is an example of what cannot be done within the current definition of desired merging behavior.

An interesting case is A10 that produces the correct result, but for the "wrong reasons". By inspecting figure 10.3, one would assume that in the first branch, the nodes a and b were swapped, and in the second the node a is updated to b, while the edit process that generated this tree in fact swapped the l-rooted *subtrees* of R in the first branch. The matching produced by 3DM corresponds to the former, incorrect, interpretation of the edits. The case is an example that it is in some cases impossible to unambiguously recover the original edit script, given only the pre and post edit trees.

10.2 Use cases

10.2.1 Use case 1: The shopping list

From the synchronization point of view the use case consists of 4 steps, corresponding to steps 4–7 in table 2.1 on page 10. The XML files for each step can be found in appendix B.

In the first step (step 4 in table 2.1) we want to combine all the separate branches of the shopping list into one integrated list. In this case, changes have been performed by Mr. Virtanen in `L2.xml` and Liisa in `L3.xml`. The common base for both branches is in `L0.xml`. Integrating the changes is done by a 3-way merge:

```
3dm --merge L0.xml L3.xml L2.xml L4.xml
```

The actual run produces an output file that is either identical to the facit file, or has the order of the inserted items swapped, depending on the order of the second and third argument (in the example the order is chosen so that the result is identical to the facit). This behavior is quite natural and expected: 3DM sees that items have been appended to the same list in both branches, and first appends the items from the first branch, then the second. To warn about this asymmetry in merging, 3DM also produces the conflict warning "Insertions/copies in both branches after the context nodes. Sequencing the insertions".

An observation was that if the copy threshold (the `--copythreshold` parameter) is set too low, the appended items will partially be considered to be copies of existing items (i.e. the `<item>`,`<price>` etc tags), leading to incorrect merging.

The next synchronization event is to bring the list suggested by the shopping service up-to-date (steps 5 and 6 in table 2.1). This is done in two phases: first the items listed in the suggested list are updated, and second, the shopping service makes its own suggestions (including adding prices and adjusting quantities). For the first phase we issue

```
3dm --merge L0.xml L4.xml L1.xml L5.xml
```

As items are appended to the end of both lists with respect to the base list, we get the same conflict warning as in the first step. Swapping the order of the branches will still produce a correct merge, but with the inserted items in a different order than the hand-made merge. The second phase is not relevant to synchronization, suffice to say that the shopping list service updates `L5.xml` and the result is `L6.xml`.

When reviewing the shopping list suggested by the service, Mrs. Virtanen wants to see which items have been changed and added by the service. In order to facilitate such a display we "merge" `L4.xml` into `L6.xml` and use the merge edit log to identify nodes in `L6.xml` that have changed from `L4.xml`. As we are only interested in obtaining the changes, we store the actual output (which is, of course, identical to `L6.xml`) in a dummy file.

```
3dm -e --merge L4.xml L6.xml L4.xml dummy.xml
```

The generated edit script is visualized in figure 10.4. As can be seen the edit script correctly identifies the changes between the two lists.

The final step is to propagate the updates of the shopping list to Liisa's PDA. Since we want to transfer only the differences we use the diff and patch functionalities of 3DM. First we generate the patch:

```
3dm --diff L3.xml L7.xml patch.xml
```

The patch is then transmitted to Liisa's PDA, where it is applied to her version of the list (i.e. `L3.xml`):

```
3dm --patch L3.xml patch.xml L7.xml
```

Diff and patch work as expected, producing the correct output on Liisa's PDA. Unfortunately there are no savings in bandwidth from transmitting the patch: using current encoding the patch is 1649 bytes whereas the target file is 1301. It can also be noted that the diff produced by the standard UNIX diff tool was 1570 bytes, so 3DM was no much worse. Differential transferring was not beneficial here, as most of the file had undergone changes, but it should prove useful when only a small percentage has changed.

All in all, the shopping list case turned out to be a textbook example of successful application of 3DM.

10.2.2 Use case 2: Related documents stay up-to-date

In this case, we have two occurrences of synchronization: synchronizing the changes in aunt Maija's profile into her home page, and synchronizing the updates to the home page (including the updated list of favorite bands) into the Book of Friends.

Merging the profile updates into the home page is accomplished with

```
3dm -e --merge personalia.xml personalia2.xml homepage.html homepage2.html
```

In this case the merge operation did not produce the same result as the facit (see appendix C). The merge operation failed to update the telephone number as well as the zip code and city of her address. In addition, the linked word "Peräseinäjoki" in the sentence "I work as a teacher at the elementary school in Peräseinäjoki" was somewhat unexpectedly updated to "Takaseinäjoki" (i.e. her new city of residence).

On the other hand, the result of the merge was rather expected if one understands the basic principles of 3DM. The telephone number, city and zip are not separate nodes in the home page, but rather occur inside a text node or as a node that is a combination of two (i.e. the city and zip are in the same text node on the home page). Correct merging is not even in theory possible with 3DM, the best imaginable result would roughly have been to have the city and zip field updated to the new city (or zip code) and the sentence citing the phone number updated to just the new number.

Of course, one cannot expect everyone that uses 3DM to be an expert. Luckily, we can use the information from the conflict log as well as merge logs to aid the user. The merging generates two warnings in the conflict log stating that there were modifications in a deleted subtree. In other words, some modifications did not make it to the merged output; in this case the update of phone number and zip code. By generating a merge log we can hilight the changes that the merge introduced to the homepage, and spot the faulty update of the city name.

Aunt Maija thus has to update her address information manually, having been warned by the merge tool about updates in the profile being ignored. At the same time she updates the list of her favorite music.

Some days later when Mr. Virtanen wants to view the page on aunt Maija in the Book of Friends, the system notices that one of the sources for the BoF, i.e. aunt Maija's homepage, has changed, and executes the synchronization update:

```
3dm --merge homepage.html homepage3.html maija-i.html maija-i2.html
```

In this case the synchronization succeeds almost perfectly. The address fields as well as the list of favorite bands is correctly updated. The phone number is not updated, once again because it does not occur as a separate text node. Note that the fields of the address occur as separate nodes, and the entire `<ul>` list structure was directly copied from the home page to the original BoF. Thus, where we have similar structure, we have successful merge. Also note that the lost update of the telephone number yields a conflict warning, alerting the user of the problem.

Summing up, we begin to see some of the limitations of 3DM in this case: The data being updated needs to be structured in a similar manner in both branches, in order to identify common parts. “Hiding” data from 3DM inside nodes will not yield the desired result. Surprising changes can occur, if some other parts than the “right” match the changed data. The problems in this case could have been remedied by structuring the address information from the profile, for instance by adding extra (invisible) tags around the zip, city and telephone number in `homepage.html`.

Avoiding sporadic unwanted updates, as the update of the city name in the first step, could perhaps be avoided by requiring that small matches occur near each other, as a cluster. If single small node seems to match a node far away from the original location, it is probably not a match, but if an entire cluster of nodes seem to have been moved to that location, the match is more likely to be correct.

10.2.3 Use case 3: Updating an annotated illustration in a document

The use case consists of a singe synchronization operation: a 3-way merge of the draft drawing (`basefig.svg`), the final drawing (`fullfig.svg`) and the chapter from the User's Guide (`modem.html`):

```
3dm --merge basefig.svg fullfig.svg modem.html final.html
```

The result of the merging was identical to the facit. Note that the figure in the User's Guide has had an extra layer added which contains the annotation made by Mr. Virtanen, so the result is indeed the merge of two drawings!

The large number of `<path>` tags with attribute values of several hundred tokens slowed down the execution of 3DM noticeably (the execution took approximately 40 s on a 1 GHz Athlon, compared to less than 5 s for the other cases), as it had to calculate the of q-gram distances between almost every path node in the base and updated figures.

By inspecting the matching, it was observed that a some “false” copies of SVG paths had been detected. Due to the different matching types (the “false” copies were not content matched) they did not interfere with the merging. By using a higher value for the copy threshold these false copies are eliminated entirely.

10.2.4 Use case 4: Merging changes from several authors

As there are three branches of the original document, we need to perform the merge in two stages: merging the base document and two of the branches, and subsequently merge the base, the output of the first stage and the third branch. In fact, using this method, any number of branches, i.e. an n-way merge, can be performed.

We start by merging the branch created by Mr. Ollila (`branch1.xml`) and the branch by Mr. Simola (`branch2.xml`).

```
3dm --merge base.xml branch1.xml branch2.xml mergeb12.xml
```

Comparing to the facit, we see that there are some differences in the document meta data: the fields for document creation, editing cycles and document statistics do not match those of the facit. OpenOffice automatically updated these fields in both branches, as well as in the facit document, and quite naturally, they cannot be merged as such. Any other deviations from the facit did not occur. Especially note how the change of the “Standard” paragraph style in branch 1 was successfully incorporated in the merge — a fine demonstration of the possibilities of structural merging indeed.

A phenomenon that occurred when producing the facit by hand using OpenOffice was that it inserted changes into the document although none had been made, in this case additional attributes for the "Standard" style and an empty text span tag. Another example was that the style name of the itemized list at the end of the document was suddenly changed from "P2" in the base document to "P3" in branch 1.

In stage two, we expect conflicts to occur: The section entitled "Cannot create lockfile. Sorry" was moved to different locations in branch 2 and 3, and there were modifications in the section entitled "For Dial-in I Keep Getting..." in branch 3, whereas the section was deleted in branch 2. Running

```
3dm --merge base.xml mergeb12.xml branch3-cf.xml dummy.xml
```

indeed generates conflicts: one sequencing conflict for the moved paragraph, and a conflict warning for the deleted section (in addition we get the same conflicts on the meta data that occurred in the first stage). We approve the section move made in branch 2 by moving the section back to its original location in branch 3. Executing

```
3dm --merge base.xml mergeb12.xml branch3.xml mergeb123.xml
```

yields the correct merge of all the branches (excluding the tree meta data). A merge log for checking the changes from the base version can be obtained as demonstrated in use case 1.

In conclusion, this case was very successful, and showed that 3DM can be used for merging "real-world" formatted documents generated by a industry-strength word processor. The only glitches were fields that were *a priori* unmergable, and did not affect the quality of the output.

10.2.5 Use case 5: Maintaining several versions of a web page

We merge the changes to the PDA version (files `ccs-simple.html` and `ccs-simple-2.html`) into the full version (`ccs-complex.html`) thusly:

```
3dm -merge ccs-simple.html ccs-simple-2.html ccs-complex.html ccs-complex-2.html
```

Comparing to the facit there are two relevant differences (the third is that the CVS `$Id$` string does not match, which was expected). Instead of enclosing the names and titles ("Jussi Virtanen *(chairman)"* and "Anna-Kaisa Pokkanaama *(secretary)"*) in the `<font>` tags, the titles (including the enclosing `<i>` tag) were inserted after the `<font>` tags. By inspecting a part of the matching by 3DM we see why this occurs (see figure 10.5).

Below the list items (i.e the `<li>` tags) for the chairman and the secretary (only the matching for the chairman is showed in the figure) 3DM sees two insertions and a move: in the first branch the `<i>` tag is inserted, in the second the `<font>` tag is inserted and the name of the person moved. Merging using this matching does not yield the result presented in the facit. On the other hand the matching seems quite reasonable.

The conclusion that can be drawn from this is that modifications from both branches that occur close to another may yield undesired results. In this case the result was acceptable (the inserts were sequenced as name - title, not title-name), but this need not be the case. As in the other use

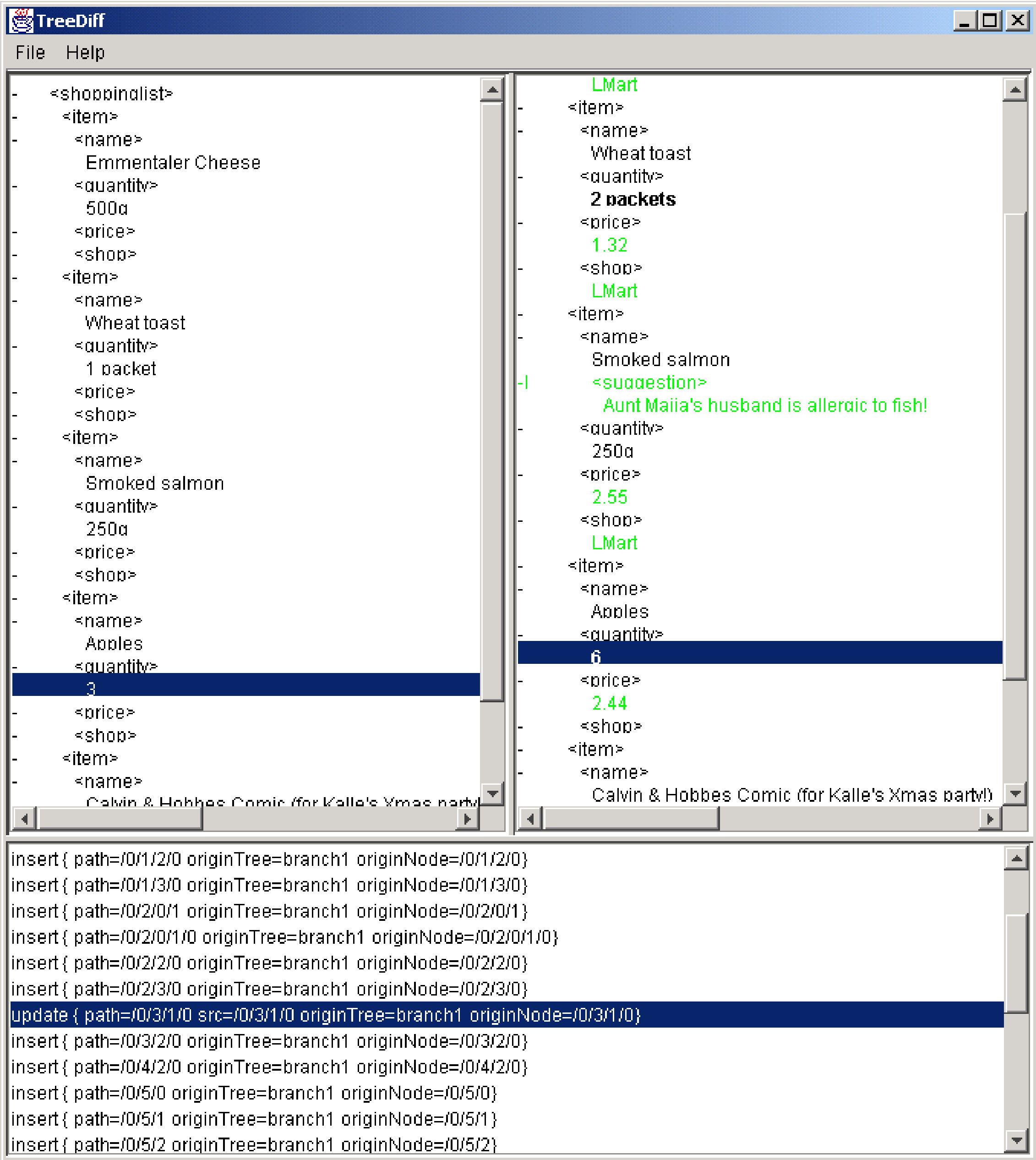
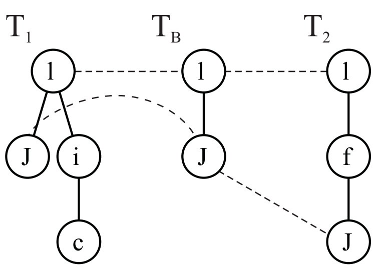


Figure 10.4: Visualization of edits between L4 and L6.



Label	Name/Text
l	
J	Jussi Virtanen
i	<i>
c	chairman

Figure 10.5: Part of the matching from the web page use case.

cases conflict warnings, in this case about sequencing inserts, gives us a warning that the result may not be what we expected.

However, all in all the case was quite successful (all the other edits were correctly propagated). It also quite successfully demonstrates a scenario that can be deemed highly relevant for web content management for different types of devices.

10.3 Conclusions of the evaluation

On the whole, 3DM performed quite well in the evaluation. If we also assume that the user is familiar with the basic limitations of the tool (such as not merging data inside nodes), we can even say that 3DM performed very well — what was supposed to work worked, and in most cases it was easy to understand why some things didn't. Also, the user should be much aided by the conflict and merge logging facilities of the tool (with the proper user interface, of course).

The merging algorithm exhibited very few problems, things that could be improved are mainly issues regarding node context locking policies. Quite surprisingly, the heuristic matcher also performed very well, exhibiting only a few non well-formed matchings. This observation is quite interesting considering all effort that has been put into producing optimal tree matchers.

The main cause of problems turned out to be integrating edits in different branches that are close to another in a parent-child sense, giving rise to the notion of some sort of context locking along the parent-child axis in addition to the context locking inside child lists.

In the use cases there occurred no copying of structures, so unfortunately we cannot draw any conclusions of how well 3DM handles copies in practice. The fact that no copying occurred rises the question if there is any need for 3DM to detect copies at all. We also note that by removing the copy functionality, the risk of falsely detected copies disappears.

Chapter 11

Conclusions and Future Work

In the beginning of the thesis we presented five detailed use cases which demonstrate the need for a synchronizing tool for tree-structured data. In chapter we3 saw that in order to perform the synchronization tasks presented in the use cases we need an algorithm capable of 3-way merging of ordered trees. From the review of existing work (chapter 4) we concluded that no such algorithm or tool currently exist, and hence we needed to design, and, for the purpose of verifying the functionality, implement, an algorithm for performing 3-way merging. As the use cases were elaborated to include the synchronization events with corresponding sample data we were able, in chapter 10, to verify the operation of the 3DM tool against the cases.

The merging algorithm is based on the definition of a desired merging behavior. In chapters 10 and 5 we derived the merging behavior from the use cases and the characteristics of the ubicomp environment they were set in. To further explore the space of possible merging situations some 40 small merging cases were written. In chapter 5, we contribute to existing research by presenting an exact and concise definition of 3-way tree merging behavior that is suitable for algorithmic implementation. The definition is based on the the *natural matching* model for matching trees that uses typed node matches that allow us to unambiguously express insert, delete, update copy and move operations, and the idea of *node contexts*.

We based the design of the merging algorithm on the division of the problem into tree matching and merging of matched trees. Even though several tree matching algorithms exist, we chose to design one of our own in chapter 6. Our algorithm is a prototype level heuristic matching algorithm, aimed to provide a proof-of-concept for natural matchings. The advantage of our method is that the allowable matchings are not artificially constrained by the algorithm, and yet the worst-case time bound on the algorithm is better than for other existing non-heuristic approaches. The actual test runs on the use and merge cases show that the heuristic matcher performs surprisingly well.

The main contribution of the thesis is the 3-way merging algorithm presented in chapter 7. As far as the author knows this is the first published 3-way merging algorithm for ordered trees that is based on directly utilizing tree matchings. The merge algorithm correctly handles cases beyond the abilities of currently available merging tools, such as a subtree being moved in one branch, and the structure of the subtree being modified in the other branch.

The central ideas of the algorithm are the concurrent traversal of the branch trees, where matched nodes are visited simultaneously, and the method for combining child lists that contain locked sequences, corresponding to the node contexts implied by the desired merging behavior. In section7.2 we saw that the algorithm has a very good worst-case time complexity, and furthermore, argued for the general correctness of the algorithm and proved some particularly important properties, such as the preservation of the required node contexts in the merged tree.

The differencing and patching algorithms presented in chapter 8are interesting in the way they encode and reconstruct tree matchings. No attempt at constructing an optimal diff in the sense of required storage was made, however.

In chapter 9 we presented a proof-of-concept implementation of the matching, merging, differencing and patching algorithms: the 3DM XML differencing and merging tool. In addition to the basic functionality of the algorithms, the tool implements logging of the operation of the merging algorithm as well as conflicts encountered, as well as provides reasonable default conflict handling. Furthermore, the implementation is modular and can easily be extended to support other means of encoding ordered trees than XML.

The implementation was verified against the use cases in chapter 10. Overall, the tool performed very well and problems were mostly encountered where expected, i.e. when the use case required that data inside a tree node should be merged. We observed how important the logging facilities of the tool were for understanding how it operates. We concluded that a node context locking scheme along the parent-child axes (similar to the locking of contexts inside child lists) in the trees could improve stability. Furthermore, we saw that the test cases did not contain enough copying of data to draw any definite conclusions on the handling of copied data.

Current plans for future research on 3DM is divided into two areas: enhancing 3DM itself and building new software on top of it.

Research that falls into the category of enhancing 3DM include the development of an efficient and robust tree matcher, which may include awareness of the DTD in order to aid the matching process. The stability of the merging algorithm would probably benefit from the introduction of required node contexts along the parent-child axis (complementing the current required contexts inside child lists). More extensive test data should be used to validate the design decisions of the merging algorithm, possibly leading to further refinements of the algorithm.

Conflict resolution capabilities can be extended by, for instance, building an interactive merging interface for the end user. Evaluating the performance of 3DM on various input data is also important to validate the complexity analysis and see if the assumptions taken in the analysis hold for real-world data.

Building on top of the basic tool, we want to integrate 3DM in a networked environment and create a storage system where data is synchronized using 3DM. This includes specification of what synchronization policies to use, keeping track of related pieces of data that needs to be synchronized and the relation between 3DM and different data filtering operations, such as views into a database. The extensions to 3DM will be designed to operate in a ubiquitous computing environment, keeping in mind the characteristics presented in section 3.2. As we start moving from the system point of view towards the end user, user studies and usability aspects will become increasingly more important.

Moving 3DM out of its current prototype state is also an important task, although not necessarily very interesting from a research point of view. The author plans to release the source of 3DM under a public license and start an open source development project for a fully functional version of 3DM. Issues that need to be addressed in a complete implementation include those listed in section 9.5, performance issues, and the integration of an HTML parser, given the large base of HTML documents that could immediately benefit from the tool. It would also be very interesting to see

3DM integrated into version management software such as CVS, i.e. CVS with XML merging and differencing capabilities.

Bibliography

[Abo96] Abowd G. "Software engineering and programming language considerations for ubiquitous computing." ACM Computing Surveys, vol. 28 no. 4, 1996

[Apa01] Apache XML Project. "The Xerces Java XML parser." `http://xml.apache.org/xerces-j/index.html` [referred 2001/9/10]

[Ask94] Asklund U. "Identifying Conflicts During Structural Merge." Proceeding of the Nordic Workshop on Programming Environment Research '94. Lund University, 1994, pp. 231–242.

[BHL01] Barnes-Lee T, Hendler J. and Lassila O. "The Semantic Web." Scientific American, May 2001

[Berger96] Berger M. "CoNus — A CSCW system supporting synchronous, asynchronous and autonomous collaborative work." In ICDP'96 : IFIP/IEEE international conference on distributed platforms, Industrial Stream / poster session. A. Schill, O. Spaniol, C. Mittasch and C. Popien (eds.), Technische Universität Bergakademie Freiberg, Freiberg, Germany, 1996, p. 27–39.

[BSV98] Berger M, Schill A. and Völksen G. "Supporting Autonomous Work and Reintegration in Collaborative Systems." In Coordination Technology for Collaborative Applications — Organizations, Processes, and Agents; Wolfram Conen, Gustaf Neumann (Eds.), Springer, New York, 1998, ISBN 3-540-64170-X, pp. 177–198

[Berk96] Berk E. "HtmlDiff: A Differencing Tool for HTML Documents." Student Project, Princeton University, Princeton, NJ, USA. 1996

[Berl90] Berliner B. "CVS II: Parallelizing software development." Proceedings of the Winter 1990 USENIX Conference (Jan 22-26, 1990, Washington DC, USA), pp. 341–352. USENIX Association. Berkeley, CA, USA. Jan 1990.

[Btw] Between project home page. `http://www.hiit.fi/fuego/between` [referred 2001/9/12]

[Bri99] Brisan D. Statement on 1/19/1999 in the discussion forum for the XML diff and merge tool. The discussion can be accessed at `http://www.alphaworks.ibm.com/tech/xmldiffmerge` [referred 2001/9/10]

[CHKS94] Ceri S, Houtsma M, Keller A. and Samarati P. "A classification of update methods for replicated databases." Technical Report CS-TR-91-1392, Stanford University, Computer Science Department, Stanford, CA, USA. 1994.

[CRGW96] Chawathe S. S, Rajaraman A, Garcia-Molina H. and Widom J. "Change detection in hierarchically structured information." In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (June 1996, Montreal, Quebec, Canada). pp. 493–504

[CG97a] Chawathe S. S. and Garcia-Molina H. "An expressive model for comparing tree-structured data." Technical Report, Stanford University Database Group, 1997. Available at `http://www.cs.umd.edu/~chaw/pubs/bbdiff-tr.ps` [referred 2001/9/10]

[CG97b] Chawathe S. S. and Garcia-Molina H. "Meaningful Change Detection in Structured Data." In Proceedings of the 97 ACM SIGMOD international conference on Management of data (May 11 – 15, 1997, Tucson, AZ ,USA). pp. 26–37

[Cha99] Chawathe S. S. "Managing change in heterogenous autonomous databases." Ph.D. Thesis, Stanford University, 1999.

[CDH00] Chen Y-F, Douglis F and Huang H. "TopBlend: An efficient implementation of HtmlDiff in Java." Proceedings of WebNet 2000 (Oct 30 – Nov 4, 2000, San Antonio, TX, USA), G. Davies and C. Owen (eds.), pp. 88–94

[Ced93] Cederqvist P. et al. "Version Management with CVS. "Signum Support AB, Linköping, Sweden, 1993, 162 p. Available electronically from `http://www.loria.fr/~molli/cvs/doc/cvs.pdf`. [referred 2001/3/16]

[Cla99] Clark J. and DeRose S. "XML path language (XPath) version 1.0." W3C Recommendation 16 Nov 1999. Available at `http://www.w3.org/TR/xpath`

[CP00] Cohen N. H. and Purakayastha A. "Toward interoperable data synchronization with COSMOS." In Proceedings of Third IEEE Workshop on Mobile Computing Systems and Applications (December 7–8, 2000, Monterey, California), pp. 138–147

[Dea98] Dearle, A. "Towards ubiquitous environments for mobile users." IEEE Internet Computing, vol. 2, no. 1, 1998, pp. 22–32

[DA00] Dey A. K. and Abowd G. D. "Towards a better understanding of context and context-awareness." In the 2000 Conference on Human Factors in Computing Systems (CHI 2000): workshop on The What, Who, Where, When, and How of Context-Awareness. The Hague, The Netherlands, April 3, 2000.

[Gol99] Goland Y. et al. RFC 2518: "HTTP extensions for distributed authoring — WebDAV." IETF, Reston VA, USA. Feb 1999 `http://www.ietf.org/rfc/rfc2518.txt` [referred 2001/3/19]

[Gro] Groove Networks inc. `http://www.groovenetworks.com/` [referred 2001/3/16]

[Gro01a] Groove Networks inc. "User's guide for Groove." `http://www.groove.net/pdf/usersguide.pdf` [referred 2001/3/16]

[Gro01b] Groove Networks inc. "Introduction to Groove." `http://www.groovenetworks.com/pdf/introducinggroove.pdf` [referred 2001/3/16]

[GS78] Guibas L. J, Sedgewick R. "A Dichromatic Framework for Balanced Trees." Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science. 1978. pp. 8–21

[HPR89] Horwitz S, Prins J. and Reps T. "Integrating Noninterfering Versions of Programs." ACM Transactions on Programming Languages and Systems, vol. 11 no. 3, July 1989, pp. 345–387

[IAw99] IBM Alphaworks. "XMLTreeDiff Update Languages." part of the XML TreeDiff download at `http://www.alphaworks.ibm.com/tech/xmltreediff` [referred 2001/9/10]

[IAw01] IBM Alphaworks. XML TreeDiff home page. `http://www.alphaworks.ibm.com/tech/xmltreediff` [referred 2001/9/10]

[IAw01b] IBM Alphaworks. XML diff and merge tool home page. `http://www.alphaworks.ibm.com/tech/xmldiffmerge` [referred 2001/9/10]

[DeltaV] IETF Delta-V working group. `http://www.webdav.org/deltav/` [referred 2001/3/19]

[SGML] "Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)." International standard ISO 8879. International Organization for Standardization, Geneva, Switzerland, 1986

[IU97] Ishii H. and Ullmer B. "Tangible bits: Towards seamless interfaces between people, bits and atoms." In the proceedings of CHI'97 (March 22 - 27, 1997, Atlanta, GA USA), pp. 234–241

[JV92] Jacobson G and Vo K-P. "Heaviest Increasing/Common Subsequence Problems." Proceedings of the 3rd Annual Symposium of Combinatorial Pattern Matching, Vol. 64, 1992.

[Law01] Lawyer S. "Modem-HOWTO." `http://www.linuxdoc.org/HOWTO/Modem-HOWTO.html` [referred 2001/3/6]

[Lot] Lotus Development corp. `http://www.lotus.com/` [referred 2001/3/26]

[Lot00] Lotus Development corp. "Inside Notes: The Architecture of Notes and the Domino Server." Lotus Development corp, Cambridge, MA, USA, 2000, 213 p. Available at `http://doc.notes.net/uafiles.nsf/docs/inside-notes/$File/insidenotes.pdf` [referred 2001/9/10]

[Lu79] Lu S-Y, "A tree-to-tree distance and its application to cluster analysis." IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 1 no. 2, 1979, pp. 219–224

[MS99] Manning and Schütze. "Foundations of Statistical Natural Language Processing." The MIT Press. 1999.

[Meg00] Megginson D. "SAX 2.0: The Simple API for XML." May 2000. **??**

[Micro] Microsoft inc. `http://www.microsoft.com/` [referred 2001/4/11]

[MD94] Munson J. P. and Dewan P. "A flexible object merging framework." In Proceedings of the conference on Computer supported cooperative work, October 22 - 26, 1994, Chapel Hill, United States. pp 231–242

[Mye86] Myers E. W. "An $O(ND)$ Difference Algorithm and its Variations." Algorithmica vol. 1 no. 2, 1986, pp. 251–266

[Nap] Napster inc. `www.napster.com`

[Nov] Novell inc. `http://www.novell.com` [referred 2001/3/26]

[Nov98] Novell inc. "GroupWise 5.5 User's Guide for Windows 95/98/NT (32-bit)." `http://www.novell.com/documentation/lg/gw55/pdfdoc/55guide.pdf` [referred 2001/3/26]

[OAS01] Organization for the Advancement of Structured Information Standards (OASIS). "The DocBook DTD." `http://www.oasis-open.org/docbook` [referred 2001/9/10]

[OOff] OpenOffice.org. OpenOffice home page. `http://www.openoffice.org/` [referred 2001/3/6]

[OOff01a] OpenOffice.org. OpenOffice Source Overview. `http://www.openoffice.org/dev_docs/source/source_overview.html` [referred 2001/3/6]

[Palm] Palm inc. `http://www.palm.com/` [referred 2001/4/11]

[Psion] Psion plc. `http://www.psion.com/` [referred 2001/4/11]

[Sat96] Satyanarayanan M. "Mobile Information Access." IEEE Personal Communications vol. 3 no. 1, 1996

[Sun] Sun Microsystems. `http://www.sun.com` [referred 2001/3/6]

[Sun01a] Sun Microsystems. "StarOffice 5.2: Features, functions and benefits." `http://www.sun.com/software/star/staroffice/5.2/features.html` [referred 2001/3/6]

[SML] SyncML Initiative ltd. `http://www.syncml.org` [referred 2001/3/16]

[SML01a] SyncML Initiative ltd. "SyncML sync protocol, version 1.0." `http://www.syncml.org/docs/syncml_protocol_v10_20001207.pdf` [referred 2001/3/16]

[SML01b] SyncML Initiative ltd. "Building an industry-wide mobile data synchronization protocol: SyncML white paper." `http://www.syncml.org/download/whitepaper.pdf` [referred 2001/4/11]

[Tai79] Tai K-C. "The tree-to-tree correction problem", Journal of the ACM, vol. 26 no 3, 1979, pp. 422–433

[Tic85] Tichy W. F. "RCS — A System for Version Control." Software — Practice and Experience vol. 15 no. 7, July 1985, pp. 637–654.

[Ukk92] Ukkonen E. "Approximate string matching with q-grams and maximal matches." Theoretical Computer Science, vol. 92 no. 1, 1992, pp. 191–211

[Use] The Usenet. `www.usenet.org`

[WSC97] Wang T-L. J, Shasha D, Chang G. J. S. et al. "Structural matching and discovery in document databases." In Proceedings of the 97 ACM SIGMOD international conference on Management of data (May 11 - 15, 1997, Tucson, AZ, USA). pp. 560–563

[Wei91] Weiser M. "The computer for the 21st century:" Scientific American, Sep 1991, pp. 94–104

[Wei93] Weiser M. "Some computer science issues in ubiquitous computing." Communications of the ACM, vol. 36 no. 7, July 1993, pp. 75–85

[W3C] World wide web consortium (W3C). `http://www.w3c.org` [referred 2001/3/5]

[W3C01a] World wide web consortium. W3C Scalable Vector Graphics (SVG). `http://www.w3.org/Graphics/SVG/` [referred 2001/3/5]

[W3C01b] World wide web consortium. Hypertext markup language home page. `http://www.w3.org/MarkUp/` [referred 2001/3/5]

[W3C01c] World wide web consortium. Amaya home page. `http://www.w3.org/Amaya/` [referred 2001/3/5]

[W3C01e] Word wide web consortium. "Extensible Markup Language (XML) 1.0." 2^{nd} ed. W3C Recommendation 6 October 2000. T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler (eds.) `http://www.w3.org/TR/2000/REC-xml-20001006` [referred 2001/9/10]

[ZS89] [chap14] Zhang K. and Shasha D. "Simple fast algorithms for the editing distance between trees and related problems" SIAM J Computing

[ZS90] Zhang K and Shasha D. "Fast algorithms for the unit cost editing distance between trees." Journal of Algorithms, 11(4), December 1990, pp. 581–621

[ZS97] Zhang K and Shasha D. Approximate tree pattern matching. In "Pattern matching in strings, trees and arrays", A. Apostolico and Z. Galil (eds.). Oxford University Press, 1997, pp. 341–371

`$Id: thesis.lyx,v 1.109 2001/12/26 15:52:21 ctl Exp $`

Appendix A

Family Virtanen Home Scenario

ACT I: Wakeup

It is a wintry Tuesday morning at the Virtanens' flat in mid-Helsinki. The family members enjoy a soft awakening, each according to his or her schedule.

Jussi's side of the bed starts giving a light massage and at the same time, at 7 am, his favorite radio station, "Auran aallot" is silently turned on. The most important events of the day, according to the calendar, are projected on the wall, just as Jussi wants it. As the time to get up approaches, the radio is periodically interrupted by short beeps - just so Jussi won't snooze too long. Anna's side of the bed stays inanimate and silent, as she can sleep for another half an hour.

As Jussi walks into the kitchen the lights in the house follow him. The lights in the bedroom are dimmed to avoid disturbing Anna's sleep.

The just awakened, but still very drowsy Liisa tries to shield her eyes from the light by pulling the blanket over her head. It's 7 am. The latest hit by "Darude" is turned on to the max. She throws the pillow at that dreadful thing, that so brutally tries to wake her up. It keeps quiet for another 5 minutes, and then starts blasting out the newest of "MaXXX", which puts a definite end to her attempts at going back to sleep. At least the system knows that, unlike her father, Liisa hates to be flooded with news in the morning, and thus keeps the wall screen blank. Some high-tempo music is all Liisa needs to get up.

Liisa notices that the bathroom has become available (7:15), and quickly gets out of bed, before somebody else gets a chance to occupy it.

At 7:20 the lights in Kalle's room start to brighten. He gently awakens to the sound of the "Kenguru loikka". His PDA, "Buddy", is blinking, indicating that a message has arrived. Kalle stretches his legs and grabs his Buddy. He ponders for a while, and then opens the message. It's today's Christmas greeting, a short Donald Duck cartoon

ACT II: Breakfast

As the first one who got up, Jussi enters the kitchen at 7:15. It is his turn to prepare breakfast, assisted by the eFridge, which advises him how to fry the eggs so they won't get burned. Fried eggs on Tuesdays are a tradition in the Virtanen family.

The eFridge also suggests heating the stove to make some porridge, as well as turning on the coffee brewer. The sweet smell of newly baked bread spreads trough the house as dad opens the Breadmaker.

Jussi fetches milk, butter, cheese, baloney and some fresh juice from the fridge. Anna has also made it out of bed, and helps him in laying out the table.

Some twenty minutes later Kalle sits down at the breakfast table. The Virtanens prefer to eat their breakfast in peace and quietness, so the kitchen is set to be calm and relaxing. The only disturbances are the occasional reminders when the breakfast is ready, or when it's time to get going.

As the last person having breakfast Liisa puts the food back into the fridge. The fridge notices that they're out of Orange juice, and adds it to the list of suggested items to purchase.

ACT III: Departure

A popular aid in dressing is the virtual mirror. When standing in front of it, it reflects your picture, just like an ordinary mirror. But you can also alter the reflected image - for instance make it wear some other clothes, or try a new hair style. Dad usually checks his outfit virtually before putting on the real one, it's much faster than trying the conventional way. Anna also makes frequent use of the mirror: she likes to try out new clothes from the ads before she goes to buy them.

After breakfast, Jussi wearing his old beloved bathrobe, decides on today's dress in front of the mirror. Having a particularly bad sense of colors, he also lets the mirror give him some advice on that point. The mirror also conveniently reminds him about today's meeting, and suggests something a little more formal. Unfortunately his favorite shirt is in the wash, so he'll have to settle for some other, the mirror advises. And, as it's raining, an umbrella would be good to take with you.

Meanwhile, the PDAs of the family members quietly retrieve data that will be needed during the day. For instance, just before departing, today's paper is downloaded into Jussi's PDA, so that he can read it on the bus.

The PDAs also check that nothing important is forgotten when leaving the house.

Liisa notices that her math textbook is missing and starts looking for it. At first she doesn't find it, and mentally prepares herself to use the e-version, but then she remembers to consult the Lost&Found service. According to it, the last observation of the book was on the floor in front of the couch in the living room, which is exactly where she finds it.

Both Kalle's Buddy and Liisa's Fred show the time until they have to leave for school. If Liisa or Kalle start running late, their electronic friends try to hurry them up.

In general, however, all devices stay quiet unless there's an urgent need to call somebody's attention, as nobody likes to be surrounded by unnecessary beeps and flashes.

Nevertheless, Jussi hates being reminded of things: he thinks it's far to stressful. Thus, he has usually disabled all reminders. Today, however, he must absolutely be on time, and has therefore set his PDA to alert him 5 minutes before the bus arrives at the stop. (5 minutes is enough to get to the stop)

A soft humming at the door indicates that the driver taking the children to school has arrived. Anna and some of her friends have put together a car pool for driving their children.

Anna remembers that it's shortly her turn to be the driver, so she opens up her calendar, as well as the booking list for the car pool, and picks a suitable day.

Just before leaving, Liisa quickly checks her dressing in the virtual mirror, and consults it about the color co-ordination. "Against all known rules", the mirror answers. "Awesome!" says Liisa, and runs off.

Mom leaves for work using the family's car.

Kalle's teacher sent a reminder regarding the school Christmas party and the 5-euro-present present she wishes that every pupil should bring to the party. While on the bus, Jussi notices the incoming message, and adds a suitable present to the shopping list.

ACT IV: At Work

Everybody gets to school or work on time, making for an excellent start of the day.

During the lunch break Liisa suddenly starts thinking that it was a long time since they had any apples at home. She adds some to the shopping list.

At around 3 o'clock in the afternoon Anna remembers that she should confirm the time of aunt Maija's visit this evening. She opens up the home calendar and calls aunt Maija. Both are able to see each others busy-hours, so finding suitable time is easy.

They decide on dinner at the Virtanens at 7 pm. Anna offers to pick Maija up on her way home.

The discussion is interrupted by an incoming call to Anna. Luckily they had already agreed on the time, so they say quick bye-byes and Anna enters aunt Maija's visit into the home calendar. Once the entry is there, it is sent to all family members.

When Jussi receives the notification of aunt Maija's visit, he remembers the rather messy state the flat is in. He checks if there are any cleaners available. It turns out that their neighbors have ordered a cleaner for this morning, and that the cleaner is available for two hours after that. He quickly makes a reservation, getting a special discount since it suits the cleaner's timetable so conveniently.

At the same time the slightly neurotic Jussi checks on the house, especially that the stove isn't on.

The new calendar entry about aunt Maija's visit awakens the shopping list service. It suggests some modifications to the shopping list (such as increased amounts due to the visit). It also checks the food selection against the preferences and possible allergies of the guests. In this case, it's really useful, since aunt Maija is a vegetarian, and her husband is allergic to fish.

Anna reviews the changes suggested by the service and adds some more vegetables and a cake.

Anna can easily interleave the home-related issues with her work, it's only a matter of switching between the home and work state on her workstation. When in the home state, she has access to all the home data, just as if she were there herself. If an urgent work issue arises she can easily switch to the work state, and then go back to the home state later to finish whatever she was doing.

The shopping list service also monitors the prices in different supermarkets, and takes this into account when making suggestions. A recipe service is also integrated, which can be used to select the course as well as to determine the amounts of food that is needed. The shopping list service is also connected to the storage service of the house. So, for instance, if there's no toilet paper left, this can also be put on the list of suggested items.

ACT V: The Children returning home

Liisa and Kalle's parents support the notion of an activating upbringing, and thus, from time to time, give Kalle and Liisa some small tasks to carry out. Today Anna suggests that Liisa could do the shopping on her way home. Liisa agrees after some bargaining involving a visit to a hamburger bar.

Kalle once again has lost his Buddy. When arriving home he cannot get in without it, so she has to ring the bell, contacting Anna at work, and ask her to let him in. Anna can see who is at the door on her screen, so she knows who she is letting in. At the same time, she checks on Kalle's Buddy using the Lost&Found service. The service tells her that Kalle has left it at his friend Ville. Anna sends a message to Ville's parents, asking them to put Kalle's Buddy into Ville's backpack the next day. There's no need to worry about someone misusing the Buddy - it locks itself in the hands of wrong people.

"Maybe I should get Kalle an intelligent wristwatch instead", Anna ponders. "At least he won't lose it so easily"

When Kalle gets into the house, a cheery video mail from Anna is waiting for him on the fridge. "Hi, little guy! If you're hungry you just grab these sausages!"

Jussi notices that the stove has been turned on, and phones home. He is calmed down by Kalle who explains that he is just boiling some sausages.

ACT VI: Jussi and Anna arrive home.

Jussi arrives at home around 5 pm, and starts preparing the dinner as soon as Liisa gets back from the supermarket. Preparing dinner is easy with aid of the eChef food advisor, so Jussi is able to do some work at the same time, using a terminal in the kitchen. There are just some documents that have to be reviewed for tomorrow but, luckily, nothing that would require phoning a video call to work. Jussi would hate to have his co-workers see the rather messy kitchen.

At 5 pm Anna's working day ends. She says goodbye to her friends at work, drives to aunt Maija, and picks her up. Everything is fine until halfway home, when the car suddenly breaks down. Rats.

Anna sends a request for service to the nearest car service shop. Luckily, all services are location-aware nowadays, so the mechanics know exactly where to go. There's no time to wait, so Anna requests that the mended car is to be driven home, and then she uses her PDA to locate the fastest bus route home. Unfortunately, taking the bus turns out to be too slow, so they order a taxi instead.

When riding the taxi Anna's PDA is able to tell the estimated time of arrival at home. She phones home to Jussi, telling him that they will be about 25 minutes late, given the current route and state of traffic. From now on Jussi is also able to monitor the location of the taxi.

ACT VII: Preparing for the visit

The instructions for making food change due to the guests running late. The oven lowers the temperature slightly so that the broccoli pie will be done some 20 minutes later than before. Jussi is also advised to put the lettuce in the fridge. At least there now is enough time to lay out the table nicely.

Jussi prepares for the visit by checking aunt Maija and her husband's infofiles. Aunt Maija's file has a note about taking that awful 5-kilo crystal vase she gave out of storage and putting it on the dinner table, as well a list of her favorite music. Jussi selects some of the tunes for playback later that night. He also sets the mood in the house to "cozy". All set for the guests!

ACT VIII: The visit

The guests arrive at half past seven, welcomed by Jussi and some soft background music.

The mandatory greetings are soon accounted for, as everybody is very hungry. Aunt Maija seems to be very happy about finding her vase at the dinner table - maybe even too happy, is she maybe suspicious?

The food is excellent. During the conversation over dinner they notice that a new episode of their favorite soap opera, "Sulautetut elämät TKK:lla" ("Integrating lives at the HUT"), has appeared. They decide to watch it immediately after dinner.

They watch the episode in the dining room. It was especially interesting, and a lively conversation arises. At the home page of the series they vote for who gets to stay on the show and for increased cuts in research funding.

Liisa is not very interested in the show, and prefers to chat with her friends over the videophone. They trade pictures, compare experiences, and generally talk about whatever teenage girls talk about.

The series also fails to keep up Kalle's interest, and after some ten minutes he decides that he would rather play some video game. When walking past the door to his room you can hear the nailgun blasts.

At 10pm it's time for Kalle to go bed. The mood of the house shifts, so that the guests understand that now is the time to go home.

Maija and her husband thank the Virtanens for a lovely evening, and hope that they will have an opportunity to visit them sometimes soon.

Appendix B

Use Case 1: Shopping List

L0.xml Page 1 of 1

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Initial shopping list -->
<!-- $Id: L0.xml,v 1.1 2001/03/07 17:08:52 ctl Exp $ -->
<shoppinglist>
 <item>
  <name>Emmentaler Cheese</name>
  <quantity>500g</quantity>
  <price>
  </price>
  <shop>
  </shop>
 </item>
 <item>
  <name>Wheat toast</name>
  <quantity>1 packet</quantity>
  <price>
  </price>
  <shop>
  </shop>
 </item>
 <item>
  <name>Smoked salmon</name>
  <quantity>250g</quantity>
  <price>
  </price>
  <shop>
  </shop>
 </item>
</shoppinglist>
```

L1.xml Page 1 of 1

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Computer suggests Orange juice, and fills in prices -->
<!-- $Id: L1.xml,v 1.2 2001/03/08 10:32:45 ctl Exp $ -->
<shoppinglist>
 <item>
  <name>Emmentaler Cheese</name>
  <quantity>500g</quantity>
  <price>3.12</price>
  <shop>LMart</shop>
 </item>
 <item>
  <name>Wheat toast</name>
  <quantity>1 packet</quantity>
  <price>0.66</price>
  <shop>LMart</shop>
 </item>
 <item>
  <name>Smoked salmon</name>
  <quantity>250g</quantity>
  <price>2.55</price>
  <shop>LMart</shop>
 </item>
 <item>
  <name>Orange juice</name>
  <quantity>2l</quantity>
  <price>1.96</price>
  <shop>LMart</shop>
 </item>
 <item>
  <name>Toilet paper</name>
  <quantity>1 packet</quantity>
  <price>3.80</price>
  <shop>LMart</shop>
 </item>
</shoppinglist>
```

L2.xml **Page 1 of 1**


```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Dad adds suggestion for Christmas party present -->
<!-- $Id: L2.xml,v 1.2 2001/06/15 12:58:55 ctl Exp $ -->
<shoppinglist>
  <item>
    <name>Emmentaler Cheese</name>
    <quantity>500g</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Wheat toast</name>
    <quantity>1 packet</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Smoked salmon</name>
    <quantity>250g</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Calvin &amp; Hobbes Comic (for Kalle’s Xmas party!)</name>
    <quantity>1</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
</shoppinglist>
```


L3.xml **Page 1 of 1**


```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Lisa adds an apple to the shopping list -->
<!-- $Id: L3.xml,v 1.2 2001/03/08 10:32:45 ctl Exp $ -->
<shoppinglist>
  <item>
    <name>Emmentaler Cheese</name>
    <quantity>500g</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Wheat toast</name>
    <quantity>1 packet</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Smoked salmon</name>
    <quantity>250g</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Apples</name>
    <quantity>3</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
</shoppinglist>
```


L4.xml **Page 1 of 1**


```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Mom starts reviewing the list. First Lisa and Dads lists are
     downloaded and merged (=the listbelow). The merged list
     is sent to Lisa’s & Dad’s PDA:s (as they are already online) -->
<!-- $Id: L4.xml,v 1.2 2001/06/15 12:58:55 ctl Exp $ -->
<shoppinglist>
  <item>
    <name>Emmentaler Cheese</name>
    <quantity>500g</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Wheat toast</name>
    <quantity>1 packet</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Smoked salmon</name>
    <quantity>250g</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Apples</name>
    <quantity>3</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Calvin &amp; Hobbes Comic (for Kalle’s Xmas party!)</name>
    <quantity>1</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
</shoppinglist>
```

L5.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- The computer looks up prices for Dad's and Lisas additions,
     as well as suggests new food amounts (and warns about the salomon!) -
->
<!-- $Id: L5.xml,v 1.3 2001/06/15 12:58:56 ctl Exp $ -->
<shoppinglist>
  <item>
    <name>Emmentaler Cheese</name>
    <quantity>500g</quantity>
    <price>3.12</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Wheat toast</name>
    <quantity>1 packet</quantity>
    <price>0.66</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Smoked salmon</name>
    <quantity>250g</quantity>
    <price>2.55</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Apples</name>
    <quantity>3</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Calvin &amp; Hobbes Comic (for Kalle's Xmas party!)</name>
    <quantity>1</quantity>
    <price>
    </price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Orange juice</name>
    <quantity>2l</quantity>
    <price>1.96</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Toilet paper</name>
    <quantity>1 packet</quantity>
    <price>3.80</price>
    <shop>LMart</shop>
  </item>
</shoppinglist>
```

L6.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- The computer looks up prices for Dad's and Lisas additions,
     as well as suggests new food amounts (and warns about the salomon!) -
->
<!-- $Id: L6.xml,v 1.3 2001/06/15 12:58:56 ctl Exp $ -->
<shoppinglist>
  <item>
    <name>Emmentaler Cheese</name>
    <quantity>500g</quantity>
    <price>3.12</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Wheat toast</name>
    <quantity>2 packets</quantity>
    <price>1.32</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Smoked salmon
    <suggestion>Aunt Maija's husband is allergic to fish!</suggestion>
    </name>
    <quantity>250g</quantity>
    <price>2.55</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Apples</name>
    <quantity>6</quantity>
    <price>2.44</price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Calvin &amp; Hobbes Comic (for Kalle's Xmas party!)</name>
    <quantity>1</quantity>
    <price>4.99</price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Orange juice</name>
    <quantity>2l</quantity>
    <price>1.96</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Toilet paper</name>
    <quantity>1 packet</quantity>
    <price>3.80</price>
    <shop>LMart</shop>
  </item>
</shoppinglist>
```

L7.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Mom has reviewed the computer suggestion list, and adds a cake and l
ettuce, remove the salmon
     and decides on 8 apples.This is the final list pulled to Lisa's PDA when s
he requests the
     shopping list. -->
<!-- $Id: L7.xml,v 1.3 2001/06/15 12:58:56 ctl Exp $ -->
<shoppinglist>
  <item>
    <name>Emmentaler Cheese</name>
    <quantity>500g</quantity>
    <price>3.12</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Wheat toast</name>
    <quantity>2 packets</quantity>
    <price>1.32</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Apples</name>
    <quantity>8</quantity>
    <price>3.26</price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Calvin &amp; Hobbes Comic (for Kalle's Xmas party!)</name>
    <quantity>1</quantity>
    <price>4.99</price>
    <shop>
    </shop>
  </item>
  <item>
    <name>Orange juice</name>
    <quantity>2l</quantity>
    <price>1.96</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Toilet paper</name>
    <quantity>1 packet</quantity>
    <price>3.80</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Priness Cake</name>
    <quantity>1</quantity>
    <price>9.98</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Lettuce</name>
    <quantity>1 head</quantity>
    <price>0.85</price>
    <shop>LMart</shop>
  </item>
</shoppinglist>
```

Appendix C

Use Case 2: Related Documents Stay Up-to-date

Contents

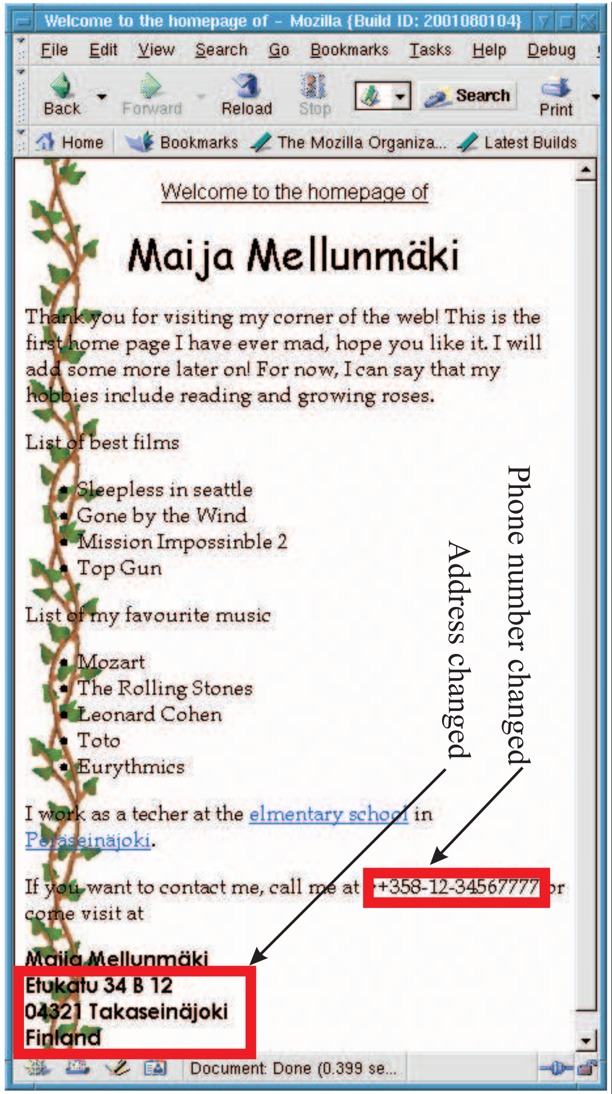
Profile, base and updated version 136
Homepage, after updated profile has been applied 137
Homepage, after favorite music has been updated 138
Base version of the Book of friends, rendered view in section 2.2 139
Updated version of the Book of friends 139
Original homepage of aunt Maija, rendered view in section 2.2 140

personalia.xml **Page 1 of 1**


```xml
<?xml version="1.0"?>
<!-- Personalia for Maija Mellunmäki -->
<!-- $Id: personalia.xml,v 1.2 2001/06/18 10:28:54 ctl Exp $ -->
<id-card>
  <name>Maija Mellunm&#228;ki</name>
  <telephone>+358-12-34567890</telephone>
  <street>Takakatu 12 B 34</street>
  <city>Per&#228;sein&#228;joki</city>
  <zip>01234</zip>
  <country>Finland</country>
  <ssn>260165-1234</ssn>
</id-card>
```


personalia2.xml **Page 1 of 1**


```xml
<?xml version="1.0"?>
<!-- Personalia for Maija Mellunmäki -->
<!-- $Id: personalia2.xml,v 1.2 2001/06/18 10:28:54 ctl Exp $ -->
<id-card>
  <name>Maija Mellunm&#228;ki</name>
  <telephone>+358-12-34567777</telephone>
  <street>Etukatu 34 B 12</street>
  <city>Takasein&#228;joki</city>
  <zip>04321</zip>
  <country>Finland</country>
  <ssn>260165-1234</ssn>
</id-card>
```

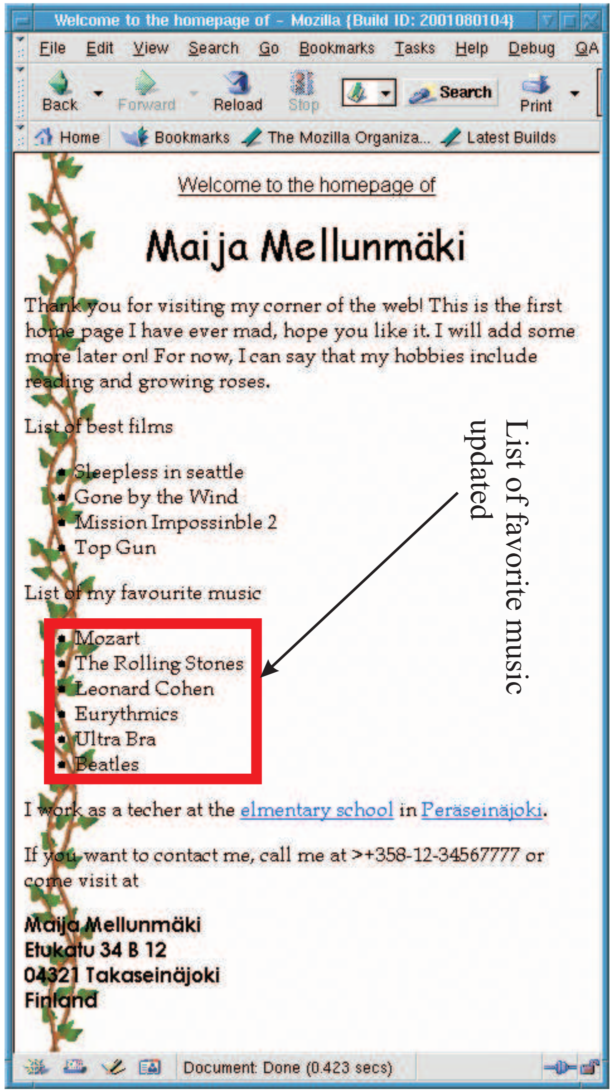


homepage2.html **Page 1 of 2**

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- Aunt Maija's homepage; after moving -> new tel and address -->
<!-- $Id: homepage2.html,v 1.1 2001/03/08 15:45:33 ctl Exp $ -->
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta name="generator" content="HTML Tidy, see www.w3.org" />
  <title>Welcome to the homepage of</title>
  <meta name="GENERATOR" content="Microsoft FrontPage 3.0" />
 </head>
 <body background="flower.jpg">
  <p align="center">
   <u>
    <font face="Arial">Welcome to the homepage of</font>
   </u>
  </p>
  <p align="center">
   <big>
    <big>
     <big>
      <big>
       <big>
        <big>
         <font face="Comic Sans MS">Maija Mellunm&#228;ki</font>
        </big>
       </big>
      </big>
     </big>
    </big>
   </big>
  </p>
  <p>
   <font face="Book Antiqua">Thank you for visiting my corner of the web!
   This is the first home page I have ever mad, hope you like it. I will
   add some more later on! For now, I can say that my hobbies include
   reading and growing roses.</font>
  </p>
  <p>
   <font face="Book Antiqua">List of best
   <blink>films</blink>
   </font>
  </p>
  <ul>
   <li>
    <font face="Book Antiqua">Sleepless in seattle</font>
   </li>
   <li>
    <font face="Book Antiqua">Gone by the Wind</font>
   </li>
   <li>
    <font face="Book Antiqua">Mission Impossinble 2</font>
   </li>
   <li>
    <font face="Book Antiqua">Top Gun</font>
   </li>
  </ul>
  <p>
   <font face="Book Antiqua">List of my favourite
   <blink>music</blink>
   </font>
  </p>
  <ul>
   <li>
    <font face="Book Antiqua">Mozart</font>
   </li>
   <li>
```

homepage2.html **Page 2 of 2**

```
    <font face="Book Antiqua">The Rolling Stones</font>
   </li>
   <li>
    <font face="Book Antiqua">Leonard Cohen</font>
   </li>
   <li>
    <font face="Book Antiqua">Toto</font>
   </li>
   <li>
    <font face="Book Antiqua">Eurythmics</font>
   </li>
  </ul>
  <p>
   <font face="Book Antiqua">I work as a techer at the
   <a href="http://elementaryschool.peraseinajoki.fi">elmentary school</a>
   in
   <a href="http://peraseinajoki.fi">Per&#228;sein&#228;joki</a>
   .</font>
  </p>
  <p>
   <font face="Book Antiqua">If you want to contact me, call me at
   &gt;+358-12-34567777 or come visit at</font>
  </p>
  <p>
   <font face="Century Gothic">Maija Mellunm&#228;ki
   <br />
   Etukatu 34 B 12
   <br />
   04321 Takasein&#228;joki
   <br />
   Finland</font>
  </p>
 </body>
</html>
```



homepage3.html **Page 1 of 2**

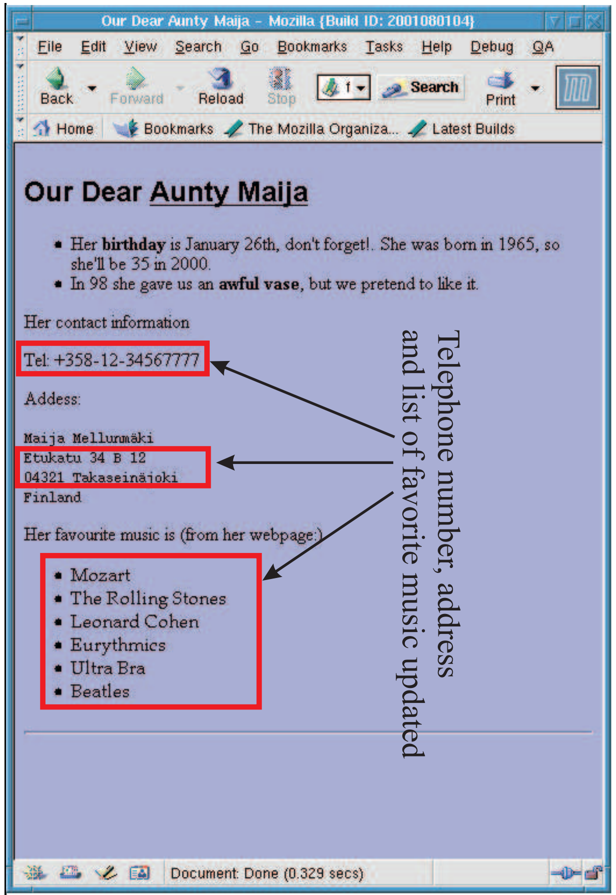
```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- Aunt Maija's homepage; favourite music changed -->
<!-- $Id: homepage3.html,v 1.1 2001/03/08 15:45:33 ctl Exp $ -->
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta name="generator" content="HTML Tidy, see www.w3.org" />
  <title>Welcome to the homepage of</title>
  <meta name="GENERATOR" content="Microsoft FrontPage 3.0" />
 </head>
 <body background="flower.jpg">
  <p align="center">
   <u>
    <font face="Arial">Welcome to the homepage of</font>
   </u>
  </p>
  <p align="center">
   <big>
    <big>
     <big>
      <big>
       <big>
        <big>
         <font face="Comic Sans MS">Maija Mellunm&#228;ki</font>
        </big>
       </big>
      </big>
     </big>
    </big>
   </big>
  </p>
  <p>
   <font face="Book Antiqua">Thank you for visiting my corner of the web!
   This is the first home page I have ever mad, hope you like it. I will
   add some more later on! For now, I can say that my hobbies include
   reading and growing roses.</font>
  </p>
  <p>
   <font face="Book Antiqua">List of best
   <blink>films</blink>
   </font>
  </p>
  <ul>
   <li>
    <font face="Book Antiqua">Sleepless in seattle</font>
   </li>
   <li>
    <font face="Book Antiqua">Gone by the Wind</font>
   </li>
   <li>
    <font face="Book Antiqua">Mission Impossinble 2</font>
   </li>
   <li>
    <font face="Book Antiqua">Top Gun</font>
   </li>
  </ul>
  <p>
   <font face="Book Antiqua">List of my favourite
   <blink>music</blink>
   </font>
  </p>
  <ul>
   <li>
    <font face="Book Antiqua">Mozart</font>
   </li>
   <li>
```

homepage3.html **Page 2 of 2**

```
    <font face="Book Antiqua">The Rolling Stones</font>
   </li>
   <li>
    <font face="Book Antiqua">Leonard Cohen</font>
   </li>
   <li>
    <font face="Book Antiqua">Eurythmics</font>
   </li>
   <li>
    <font face="Book Antiqua">Ultra Bra</font>
   </li>
   <li>
    <font face="Book Antiqua">Beatles</font>
   </li>
  </ul>
  <p>
   <font face="Book Antiqua">I work as a techer at the
   <a href="http://elementaryschool.peraseinajoki.fi">elmentary school</a>
   in
   <a href="http://peraseinajoki.fi">Per&#228;sein&#228;joki</a>
   .</font>
  </p>
  <p>
   <font face="Book Antiqua">If you want to contact me, call me at
   &gt;+358-12-34567777 or come visit at</font>
  </p>
  <p>
   <font face="Century Gothic">Maija Mellunm&#228;ki
   <br />
   Etukatu 34 B 12
   <br />
   04321 Takasein&#228;joki
   <br />
   Finland</font>
  </p>
 </body>
</html>
```

maija-i.html **Page 1 of 1**

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- Aunt Maija infopage -->
<!-- $Id: maija-i.html,v 1.1 2001/03/08 15:45:33 ctl Exp $ -->
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta name="generator" content="HTML Tidy, see www.w3.org" />
  <title>Our Dear Aunty Maija</title>
 </head>
 <body bgcolor="#9999CC">
  <h1>
   <small>
    <font face="Arial">Our Dear
    <u>Aunty Maija</u>
    </font>
   </small>
  </h1>
  <ul>
   <li>Her
   <strong>birthday</strong>
   is January 26th, don't forget!. She was born in 1965, so she'll be 35
   in 2000.</li>
   <li>In 98 she gave us an
   <strong>awful vase</strong>
   , but we pretend to like it.</li>
  </ul>
  <p>Her contact information</p>
  <p>Tel: +358-12-34567890</p>
  <p>Addess:</p>
  <p>
   <tt>Maija Mellunm&#228;ki
   <br />
   Takakatu 12 B 34
   <br />
   01234 Per&#228;sein&#228;joki
   <br />
   Finland</tt>
  </p>
  <p>Her favourite music is (from her webpage:)</p>
  <ul>
   <li>
    <font face="Book Antiqua">Mozart</font>
   </li>
   <li>
    <font face="Book Antiqua">The Rolling Stones</font>
   </li>
   <li>
    <font face="Book Antiqua">Leonard Cohen</font>
   </li>
   <li>
    <font face="Book Antiqua">Toto</font>
   </li>
   <li>
    <font face="Book Antiqua">Eurythmics</font>
   </li>
  </ul>
  <hr />
 </body>
</html>
```



maija-i2.html **Page 1 of 1**

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- Aunt Maija infopage -->
<!-- $Id: maija-i2.html,v 1.2 2001/06/18 10:28:54 ctl Exp $ -->
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta name="generator" content="HTML Tidy, see www.w3.org" />
  <title>Our Dear Aunty Maija</title>
 </head>
 <body bgcolor="#9999CC">
  <h1>
   <small>
    <font face="Arial">Our Dear
    <u>Aunty Maija</u>
    </font>
   </small>
  </h1>
  <ul>
   <li>Her
   <strong>birthday</strong>
   is January 26th, don't forget!. She was born in 1965, so she'll be 35
   in 2000.</li>
   <li>In 98 she gave us an
   <strong>awful vase</strong>
   , but we pretend to like it.</li>
  </ul>
  <p>Her contact information</p>
  <p>Tel: +358-12-34567777</p>
  <p>Addess:</p>
  <p>
   <tt>Maija Mellunm&#228;ki
   <br />
   Etukatu 34 B 12
   <br />
   04321 Takasein&#228;joki
   <br />
   Finland</tt>
  </p>
  <p>Her favourite music is (from her webpage:)</p>
  <ul>
   <li>
    <font face="Book Antiqua">Mozart</font>
   </li>
   <li>
    <font face="Book Antiqua">The Rolling Stones</font>
   </li>
   <li>
    <font face="Book Antiqua">Leonard Cohen</font>
   </li>
   <li>
    <font face="Book Antiqua">Eurythmics</font>
   </li>
   <li>
    <font face="Book Antiqua">Ultra Bra</font>
   </li>
   <li>
    <font face="Book Antiqua">Beatles</font>
   </li>
  </ul>
  <hr />
 </body>
</html>
```

homepage.html **Page 1 of 2**

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- Aunt Maija's homepage -->
<!-- $Id: homepage.html,v 1.1 2001/03/08 15:45:33 ctl Exp $ -->
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta name="generator" content="HTML Tidy, see www.w3.org" />
  <title>Welcome to the homepage of</title>
  <meta name="GENERATOR" content="Microsoft FrontPage 3.0" />
 </head>
 <body background="flower.jpg">
  <p align="center">
   <u>
    <font face="Arial">Welcome to the homepage of</font>
   </u>
  </p>
  <p align="center">
   <big>
    <big>
     <big>
      <big>
       <big>
        <big>
         <font face="Comic Sans MS">Maija Mellunm&#228;ki</font>
        </big>
       </big>
      </big>
     </big>
    </big>
   </big>
  </p>
  <p>
   <font face="Book Antiqua">Thank you for visiting my corner of the web!
   This is the first home page I have ever mad, hope you like it. I will
   add some more later on! For now, I can say that my hobbies include
   reading and growing roses.</font>
  </p>
  <p>
   <font face="Book Antiqua">List of best
   <blink>films</blink>
   </font>
  </p>
  <ul>
   <li>
    <font face="Book Antiqua">Sleepless in seattle</font>
   </li>
   <li>
    <font face="Book Antiqua">Gone by the Wind</font>
   </li>
   <li>
    <font face="Book Antiqua">Mission Impossinble 2</font>
   </li>
   <li>
    <font face="Book Antiqua">Top Gun</font>
   </li>
  </ul>
  <p>
   <font face="Book Antiqua">List of my favourite
   <blink>music</blink>
   </font>
  </p>
  <ul>
   <li>
    <font face="Book Antiqua">Mozart</font>
   </li>
   <li>
```

homepage.html **Page 2 of 2**

```
    <font face="Book Antiqua">The Rolling Stones</font>
   </li>
   <li>
    <font face="Book Antiqua">Leonard Cohen</font>
   </li>
   <li>
    <font face="Book Antiqua">Toto</font>
   </li>
   <li>
    <font face="Book Antiqua">Eurythmics</font>
   </li>
  </ul>
  <p>
   <font face="Book Antiqua">I work as a techer at the
   <a href="http://elementaryschool.peraseinajoki.fi">elmentary school</a>
   in
   <a href="http://peraseinajoki.fi">Per&#228;sein&#228;joki</a>
   .</font>
  </p>
  <p>
   <font face="Book Antiqua">If you want to contact me, call me at
   +358-12-34567890 or come visit at</font>
  </p>
  <p>
   <font face="Century Gothic">Maija Mellunm&#228;ki
   <br />
   Takakatu 12 B 34
   <br />
   01234 Per&#228;sein&#228;joki
   <br />
   Finland</font>
  </p>
 </body>
</html>
```

Appendix D

Use Case 3: Updating an Annotated Illustration

Contents

Draft modem drawing, SVG source (rendering in section 2.3) 141

Completed modem drawing, rendering and SVG source 142

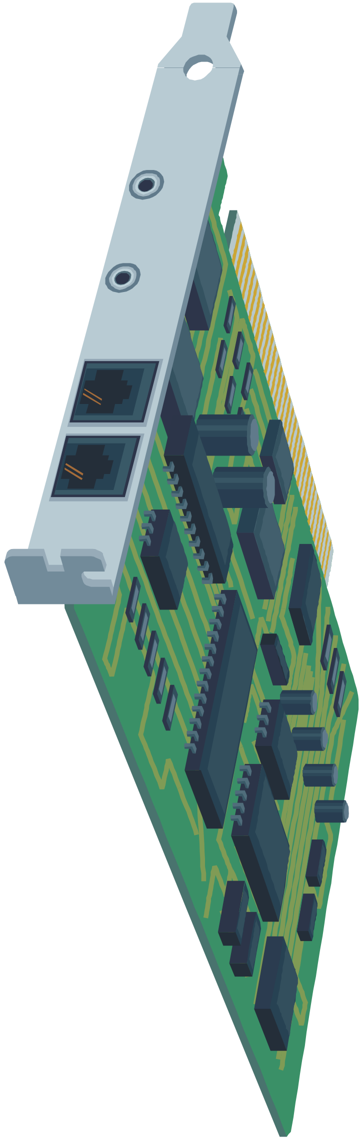
Base version of manual, XHTML source (rendering in section 2.3) 143

Manual with updated drawing, XHTML source (rendering in section 2.3) 144

The source listings (SVG and XHTML) have been shortened to occupy less space. The ellipsis (both vertical and horizontal) indicate omissions in the form of shortened paragraphs and removed `<path>` tags (the drawing consists of several hundred paths, the listing of which would hardly be informative).

basefig.svg **Page 1 of 1**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000303 Stylable//EN"
"http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-styla
ble.dtd">
<!-- Creator: CorelDRAW -->
<!-- Unfinished drawing -->
<!-- $Id: basefig.svg,v 1.2 2001/03/30 09:43:01 ctl Exp $ -->
<svg xml:space="preserve" x="-2.55184in" y="0.797028in"
width="5.10368in" height="1.59406in"
style="shape-rendering:geometricPrecision; text-rendering:geometricPreci
sion; image-rendering:optimizeQuality"
 viewBox="-2552 0 5104 1594">

<g id="Layer 1">
<path id="31680424" style="fill:#3E5F5C"
d="M-2185 744l413 -120 176 43 157 -46 -179 -40 0 -34 817 -200 712 -202 168
28 488 -141 61 -2 900 147 1024 146 0 32 -1034 423 -1345 545 -84 3 -1224 -303
-1050 -259 0 -20z" />

      .
      .
      .

<path id="31664384" style="fill:#855F2D"
d="M-433 1239l9 3 -49 78 -7 -4 47 -77z" />

<path id="31664472" style="fill:#855F2D"
d="M-411 1242l7 5 -47 76 -9 0 49 -81z" />

<path id="31664560" style="fill:#8898A7"
d="M82 1103l-38 11 0 83 -4 12 -1 11 -10 9 -5 5 -26 8 -10 0 -5 -5 -5 -8 0 -10 30 -
76 -30 -9 -45 16 -7 1 -5 5 -5 9 -4 10 -5 10 -1 12 0 141 33 5 39 -27 0 78 -72 22 0 1
36 1 12 5 10 9 8 12 4 33 8 116 -56 36 -430 -36 -5z" />

<path id="31664648" style="fill:#657786"
d="M118 1108l-41 13 0 83 -3 10 -3 10 -7 10 -8 5 -24 8 -12 0 -7 -5 -5 -7 0 -92 -42
 13 -6 2 -5 3 -5 11 -4 10 -5 10 -2 12 0 139 69 -20 0 77 -69 21 0 138 2 11 5 9 9 12
 11 3 152 -50 0 -436z" />
</g>
</svg>
```



fullfig.svg **Page 1 of 1**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000303 Stylable//EN"
"http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-styla
ble.dtd">
<!-- Creator: CorelDRAW -->
<!-- Complete modem drawing -->
<!-- $Id: fullfig.svg,v 1.1 2001/03/05 15:33:46 ctl Exp $ -->
<svg xml:space="preserve" x="-2.55184in" y="0.797028in"
width="5.10368in" height="1.59406in"
style="shape-rendering:geometricPrecision; text-rendering:geometricPreci
sion; image-rendering:optimizeQuality"
 viewBox="-2552 0 5104 1594">

<g id="Layer 1">
<path id="31680424" style="fill:#3E5F5C"
d="M-2185 744l413 -120 176 43 157 -46 -179 -40 0 -34 817 -200 712 -202 168
28 488 -141 61 -2 900 147 1024 146 0 32 -1034 423 -1345 545 -84 3 -1224 -303
-1050 -259 0 -20z" />

      .
      .
      .

<path id="31333956" style="fill:#2A3A48"
d="M1369 180l-19 -3 -17 0 -17 -4 -15 -1 -17 -4 -16 -3 -17 -3 -15 0 16 -9 19 -3 15
0 17 3 17 4 17 3 15 2 18 3 15 0 19 5 -19 3 -16 7z" />

<path id="31334044" style="fill:#1A1D26"
d="M1426 172l0 52 -46 15 0 -52 46 -15z" />

<path id="31334132" style="fill:#212D3D"
d="M1671 261l0 -51 -168 -25 -45 14 0 50 168 29 45 -17z" />

<path id="31334220" style="fill:#202233"
d="M1626 278l0 -52 -168 -27 0 50 168 29z" />

<path id="31334308" style="fill:#212D3D"
d="M1626 226l-12 -4 -10 -3 -12 0 -10 -4 -12 0 -10 -1 -10 -2 -10 0 -12 -5 -10 0 -10
 -3 -10 0 -10 -4 -10 0 -10 -1 -10 0 10 -4 12 -3 11 -4 12 -3 10 0 10 2 10 1 11 0 10 0
 10 4 10 0 12 5 10 0 10 2 10 1 12 4 10 0 12 0 10 3 11 3 -11 4 -12 1 -12 4 -10 7z"
/>

<path id="31334396" style="fill:#2A3A48"
d="M1613 219l-19 -4 -15 0 -19 -3 -15 -2 -17 -3 -16 -3 -17 -4 -15 0 8 -3 10 -5 9 -4
10 0 15 0 17 4 16 3 17 4 15 1 17 4 17 0 17 3 -19 7 -16 5z" />

<path id="31334484" style="fill:#1A1D26"
d="M1671 210l0 51 -45 17 0 -52 45 -16z" />
 </g>
</svg>
```

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- SVG graphics use case -->
<!-- $Id: modem.html,v 1.1 2001/03/05 15:33:46 ctl Exp $ -->
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta name="generator" content="HTML Tidy, see www.w3.org" />
  <title>Installing Your Modem</title>
 </head>
 <body>
  <h1>Installing Your Modem</h1>
  <p>Congratulations for buying the SuperUltra VoiceModem 56000+
  ! In this section we will show you how to install your modem,
  and configure it for use with popular telecommunications
  software.</p>
  <dl>
   <dt>NOTE:</dt>
   <dd>Before installing the modem you should ground yourself by
   touching a kitchen sink, radiator or metal plumming. Failing
   to do som may cause damage to your modem.</dd>
  </dl>
  <ol>
   <li>Turn off your computer and unplug it from the electrical
   outlet.</li>
   <li>Remove the screws from your computer's cover and then
   remove the cover. Refer to your computer manufacturer's
   manual if you need further
   <br />
   instructions.</li>
   <li>Locate an empty expansion slot. Carefully insert the
   modem into the slot.
   <br />
<!-- the table is a placeholder for the svg graphics
     REMEMEBER to change the svg dimension.
   Too bad there are no browsers capable of showing this...
   Also possibly a bug in Corel.
     -->
<!--- Svg graphic, uncomment to run 3dm
Creator: CorelDRAW
Original inlined drawing
$Id: modem.html,v 1.1 2001/03/05 15:33:46 ctl Exp $
-->
<svg xmlns="http://www.w3.org/2000/svg" xml:space="preserve"
width="15cm"
style="shape-rendering:geometricPrecision; text-rendering:geometricPreci
sion; image-rendering:optimizeQuality"
 viewBox="-2552 0 5104 1993">

<g id="Layer 1">
<g>
<path id="31680424" style="fill:#3E5F5C"
d="M-2185 744l413 -120 176 43 157 -46 -179 -40 0 -34 817 -200 712 -202 168
28 488 -141 61 -2 900 147 1024 146 0 32 -1034 423 -1345 545 -84 3 -1224 -303
-1050 -259 0 -20z" />

    .
    .
    .

<path id="31664648" style="fill:#657786"
d="M118 1108l-41 13 0 83 -3 10 -3 10 -7 10 -8 5 -24 8 -12 0 -7 -5 -5 -7 0 -92 -42
 13 -6 2 -5 3 -5 11 -4 10 -5 10 -2 12 0 139 69 -20 0 77 -69 21 0 138 2 11 5 9 9 12
 11 3 152 -50 0 -436z" />
 </g>

<path id="32310112"
```



```
style="fill:none;stroke:#DA251D;stroke-width:28"
d="M-634 943l333 80 0 392 -333 -80 0 -392z" />

<path id="31330260"
style="fill:none;stroke:#DA251D;stroke-width:28"
d="M-959 862l333 80 0 391 -333 -79 0 -392z" />

<ellipse id="32307736"
style="fill:none;stroke:#DA251D;stroke-width:28" cx="-1734"
cy="955" rx="94" ry="102" />

<ellipse id="31329996"
style="fill:none;stroke:#DA251D;stroke-width:28" cx="-1320"
cy="1061" rx="94" ry="102" />

<text x="-2495" y="1293"
style="fill:#1F1A17;font-weight:normal;font-size:167;font-family:Arial">
Line in</text>

<text x="-2503" y="1504"
style="fill:#1F1A17;font-weight:normal;font-size:167;font-family:Arial">
Line out</text>

<text x="-2194" y="1805"
style="fill:#1F1A17;font-weight:normal;font-size:167;font-family:Arial">
Phone jack</text>

<text x="-2031" y="1992"
style="fill:#1F1A17;font-weight:normal;font-size:167;font-family:Arial">
Receiver</text>

<path id="58002056" style="fill:none;stroke:#1F1A17;stroke-width:3"
d="M-1975 1244l122 0 90 -154" />

<path id="58001880" style="fill:none;stroke:#1F1A17;stroke-width:3"
d="M-1901 1471l430 0 114 -284" />

<path id="58001704" style="fill:none;stroke:#1F1A17;stroke-width:3"
d="M-845 1333l-122 423 -374 0" />

<path id="31335452" style="fill:none;stroke:#1F1A17;stroke-width:3"
d="M-479 1406l-139 561 -724 0" />
 </g>
</svg>
<!-- END: The svg outcomment ended here -->
<!--
    <table bgcolor="#7f7f7f" width="500" height="400">
     <tr>
      <td>I'm the SVG image</td>
     </tr>
    </table>
-->
   </li>
   <li>Connect the phone cord connected to the wall to the phone
   jack connector (see the figure)</li>
   <li>Connect your receiver to the receiver jack</li>
   <li>Connect the external voice source to the line in
   jack</li>
   <li>Connect the line out jack to your hifi amplifier.</li>
   <li>Put the cover back on, connect the power and turn on your
   computer</li>
  </ol>
  <h2>Configuring the serial port</h2>
  <p>In many cases, configuring will happen automatically and you
  ... under Linux:</p>
  <p>The first part (low-level configuring) is assigning it an IO
  ... details it's easy to get into trouble.</p>
```



```
  <p>The second part (high-level configuring) is assigning it a
  ... addresses):</p>
  <ul>
   <li>Plan to use more than 2 serial ports</li>
   <li>Installing a new serial port</li>
   <li>Having problems with serial port(s)</li>
  </ul>
  <p>For kernel 2.2+ you may be able to use more that 2 serial
  ... Kernels 2.2+</p>
  <p>The low-level configuring (setting the IRQ and IO address)
  ... wrong.</p>
  <p>In the Wintel world, the IO address and IRQ are called ...
  places:</p>
  <ol>
   <li>the device driver (often by running "setserial" at
   boot-time)</li>
   <li>memory registers of the serial port hardware itself</li>
  </ol>
  <p>You may watch the start-up (= boot-time) messages. They are
  ... See I/O Address &amp; IRQ: Boot-time messages.</p>
 </body>
</html>
```

final.html **Page 1 of 3**

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- SVG graphics use case -->
<!-- $Id: final.html,v 1.2 2001/06/18 14:56:50 ctl Exp $ -->
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta name="generator" content="HTML Tidy, see www.w3.org" />
    <title>Installing Your Modem</title>
  </head>
  <body>
    <h1>Installing Your Modem</h1>
    <p>Congratulations for buying the SuperUltra VoiceModem 56000+
    ! In this section we will show you how to install your modem,
    and configure it for use with popular telecommunications
    software.</p>
    <dl>
      <dt>NOTE:</dt>
      <dd>Before installing the modem you should ground yourself by
      touching a kitchen sink, radiator or metal plumming. Failing
      to do som may cause damage to your modem.</dd>
    </dl>
    <ol>
      <li>Turn off your computer and unplug it from the electrical
      outlet.</li>
      <li>Remove the screws from your computer's cover and then
      remove the cover. Refer to your computer manufacturer's
      manual if you need further
      <br />
      instructions.</li>
      <li>Locate an empty expansion slot. Carefully insert the
      modem into the slot.
      <br />
<!-- the table is a placeholder for the svg graphics
          REMEMEBER to change the svg dimension.
          Too bad there are no browsers capable of showing this...
          Also possibly a bug in Corel.
          -->
<!--- Svg graphic, uncomment to run 3dm
Creator: CorelDRAW
Original inlined drawing
$Id: final.html,v 1.2 2001/06/18 14:56:50 ctl Exp $
-->
<svg xmlns="http://www.w3.org/2000/svg" xml:space="preserve"
width="15cm"
style="shape-rendering:geometricPrecision; text-rendering:geometricPreci
sion; image-rendering:optimizeQuality"
 viewBox="-2552 0 5104 1993">

<g id="Layer 1">
<g>
<path id="31680424" style="fill:#3E5F5C"
d="M-2185 744l413 -120 176 43 157 -46 -179 -40 0 -34 817 -200 712 -202 168
28 488 -141 61 -2 900 147 1024 146 0 32 -1034 423 -1345 545 -84 3 -1224 -303
-1050 -259 0 -20z" />

      .
      .
      .

<path id="31334396" style="fill:#2A3A48"
d="M1613 219l-19 -4 -15 0 -19 -3 -15 -2 -17 -3 -16 -3 -17 -4 -15 0 8 -3 10 -5 9 -4
10 0 15 0 17 4 16 3 17 4 15 1 17 4 17 0 17 3 -19 7 -16 5z" />

<path id="31334484" style="fill:#1A1D26"
d="M1671 210l0 51 -45 17 0 -52 45 -16z" />
  </g>
```

final.html **Page 2 of 3**

```
<path id="32310112"
style="fill:none;stroke:#DA251D;stroke-width:28"
d="M-634 943l333 80 0 392 -333 -80 0 -392z" />

<path id="31330260"
style="fill:none;stroke:#DA251D;stroke-width:28"
d="M-959 862l333 80 0 391 -333 -79 0 -392z" />

<ellipse id="32307736"
style="fill:none;stroke:#DA251D;stroke-width:28" cx="-1734"
cy="955" rx="94" ry="102" />

<ellipse id="31329996"
style="fill:none;stroke:#DA251D;stroke-width:28" cx="-1320"
cy="1061" rx="94" ry="102" />

<text x="-2495" y="1293"
style="fill:#1F1A17;font-weight:normal;font-size:167;font-family:Arial">
Line in</text>

<text x="-2503" y="1504"
style="fill:#1F1A17;font-weight:normal;font-size:167;font-family:Arial">
Line out</text>

<text x="-2194" y="1805"
style="fill:#1F1A17;font-weight:normal;font-size:167;font-family:Arial">
Phone jack</text>

<text x="-2031" y="1992"
style="fill:#1F1A17;font-weight:normal;font-size:167;font-family:Arial">
Receiver</text>

<path id="58002056" style="fill:none;stroke:#1F1A17;stroke-width:3"
d="M-1975 1244l122 0 90 -154" />

<path id="58001880" style="fill:none;stroke:#1F1A17;stroke-width:3"
d="M-1901 1471l430 0 114 -284" />

<path id="58001704" style="fill:none;stroke:#1F1A17;stroke-width:3"
d="M-845 1333l-122 423 -374 0" />

<path id="31335452" style="fill:none;stroke:#1F1A17;stroke-width:3"
d="M-479 1406l-139 561 -724 0" />
 </g>
</svg>
<!-- END: The svg outcomment ended here -->
<!--
        <table bgcolor="#7f7f7f" width="500" height="400">
          <tr>
            <td>I'm the SVG image</td>
          </tr>
        </table>
-->
      </li>
      <li>Connect the phone cord connected to the wall to the phone
      jack connector (see the figure)</li>
      <li>Connect your receiver to the receiver jack</li>
      <li>Connect the external voice source to the line in
      jack</li>
      <li>Connect the line out jack to your hifi amplifier.</li>
      <li>Put the cover back on, connect the power and turn on your
      computer</li>
    </ol>
    <h2>Configuring the serial port</h2>
    <p>In many cases, configuring will happen automatically and you
    ... under Linux:</p>
```

final.html **Page 3 of 3**

```
    <p>The first part (low-level configuring) is assigning it an IO
    ... details it's easy to get into trouble.</p>
    <p>The second part (high-level configuring) is assigning it a
    ... addresses):</p>
    <ul>
      <li>Plan to use more than 2 serial ports</li>
      <li>Installing a new serial port</li>
      <li>Having problems with serial port(s)</li>
    </ul>
    <p>For kernel 2.2+ you may be able to use more that 2 serial
    ... Kernels 2.2+</p>
    <p>The low-level configuring (setting the IRQ and IO address)
    ... wrong.</p>
    <p>In the Wintel world, the IO address and IRQ are called ...
    places:</p>
    <ol>
      <li>the device driver (often by running "setserial" at
      boot-time)</li>
      <li>memory registers of the serial port hardware itself</li>
    </ol>
    <p>You may watch the start-up (= boot-time) messages. They are
    ... See I/O Address &amp; IRQ: Boot-time messages.</p>
  </body>
</html>
```

Appendix E

Use Case 4: Document Review

The base version is shown on pages 146–148. A manually generated version showing the changes made by each author follows on pages 149–153. The desired merged (manually generated) version, after conflicts have been resolved, is shown on pages 154–156.

In the version showing the changes each author has his own color: changes by Mr. Ollila are shown in green, changes by Mr. Simola in blue and the changes by Ms. Jaatinen are in red. Each edit is identified by a number in the margin of the manual. The edits are explained in the table below.

Edit	Edits by Mr. Ollila	Edits by Mr. Simola	Edits by Ms. Jaatinen
1	Style of standard text paragraph changed to include indentation and spacing.	-	-
2	-	A space was removed	-
3	-	-	Text deleted
4	"--" replaced with "-"	-	-
5	The phrase "no response" in italics, errorneous starting chars of lines removed	-	-
6	Style of text changed to heading	-	-
7	-	Text deleted	-
8	-	-	Paragraph moved here
9	-	Formatting of text	-
10	-	Paragraph deleted	Text inserted
11	-	-	Formatting of text
12	Text formatted as a single paragraph, instead of many.	-	-
13	-	-	Text formatted
14	-	-	Paragraph moved away, see edit no 8.
15	-	-	Paragraph deleted
16	-	-	Paragraph deleted
17	-	Paragraph moved here	
18		Paragraph moved away, see edit 17	Paragraph moved away, see edit 20
19	-	Correction of misspelling	-
20	-	-	Paragraph moved here

Troubleshooting

My Modem is Physically There but Can’t be Found

The error messages could be something like "No modem detected", "Modem not responding", or (strange) "You are already online" (from Minicom). If you have installed an internal modem (serial port is builtin) or are using an external one and don’t know what serial port it’s connected to then the problem is to find the serial port. See My Serial Port is Physically There but Can’t be Found. This section is about finding out which serial port has the modem on it.

There’s a program that looks for modems on commonly used serial ports called "wvdialconf". Just type "wvdialconf <a−new−file−name>". It will create the new file as a configuration file but you don’t need this file unless you are going to use "wvdial" for dialing. See What is wvdialconf ? Unfortunately, if your modem is in "online data" mode, wvdialconf will report "No modem detected" See No response to AT

Your problem could be due to a winmodem (or the like) which usually can’t be used with Linux. See Software−based Modems (winmodems). The "setserial program may be used to detect serial ports but will not detect modems on them. Thus "wvdialconf" is best to try first.

Another way try to find out if there’s a modem on a port is to start "minicom" on the port (after first setting up minicom for the correct serial port −−you will need to save the setup and then exit minicom and start it again). Then type "AT" and you should see OK (or 0 if it’s set for "digit result codes"). The results may be:

- No response. See No response to AT
- It takes many seconds to get an expected truncated response (including only the cursor moving down one line). See Extremely Slow: Text appears on the screen slowly after long delays
- * Some strange characters appear but they are not in response to AT. This likely means that your modem is still connected to something at the other end of the phone line which is sending some cryptic packets or the like.

No response to AT

The modem should send you "OK" in response to your "AT" which you type to the modem (using minicom or the like). If you don’t see "OK" (and in most cases don’t even see the "AT" you typed either) then the modem is not responding (often because what you type doesn’t even get to the modem).

A common cause is that there is no modem on the serial port you are typing to. For the case of an internal modem, that serial port likely doesn’t exist either. That’s because the PnP modem card (which has a built−in serial port) has either not been configured (by isapnp or the like) or has been configured incorrectly. See My Serial Port is Physically There but Can’t be Found.

If what you type is really getting thru to a modem, then the lack of response could be due to the modem being in "online data" mode where it can’t accept any AT commands. You may have been using the modem and then abruptly disconnected (such as killing the process with signal 9). In that case your modem did not get reset to "command mode" where it can interact to AT commands. Thus the message from minicom "You are already online. Hangup first." Well, you are sort of online but you are may not be connected to anything over the phone line. Wvdial will report "modem not responding" for the same situation.

To fix this as a last resort you could reboot the computer. Another way to try to fix this is to send +++ to the modem to tell it to escape back to "command mode" from "online data mode". On both sides of the +++ sequence there must be about 1 second of delay (nothing sent during "guard time"). This may not work if another process is using the modem since the +++ sequence could wind up with other characters inserted in between them or after the +++ (during the guard time). Ironically, even if the modem line is idle, typing an unexpected +++ is likely to set off an exchange of control packets (that you never see) that will violate the required guard time so that the +++ doesn’t do what you wanted. +++ is usually in the string that is named "hangup string" so if you command minicom (or the like) to hangup it might work. Another way to do this is to just exit minicom and then run minicom again.

"Modem is busy"

What this means depends on what program sent it. The modem could actually be in use (busy). Another cause reported for the SuSE distribution is that there may be two serial drivers present instead of one. One driver was built into the kernel and the second was a module.

In kppp, this message is sent when an attempt to get/set the serial port parameters fails. These parameters are the one you see if you give the "stty −a" command. It’s similar to the "Input/output error" one may get when trying to use "stty −F /dev/ttySx". Although the port is already opened when you see this message, I think it’s possible to open a non−existent device (or one with the wrong IRQ or IO address). Now getting certain port parameters (such as the speed) means communicating with the serial port hardware. So this error message may mean that there is no serial port there (or that the modem’s serial port has an incorrect (or no) IRQ or I0 address. It could be a wrong ttySx number. If /dev/modem is used it should be linked to the correct ttySx. In these cases the error message should have said: "Modem can’t be found" which really means that it’s serial port (often built into the modem) can’t be found.

I can’t get near 56k on my 56k modem

There must be very low noise on the line for it to work at even close to 56k. Some phone lines are so bad that the speeds obtainable are much slower than 56k (like 28.8k or even slower). Sometimes extension phones connected to the same line can cause problems. To test this you might connect your modem directly at the point where the telephone line enters the building with the feeds for everything else on that line disconnected (if others can tolerate such a test).

Uploading (downloading) files is broken/slow

Flow control (both at your PC and/or modem−to−modem) may not be enabled. For the uploading case: If you have set a high DTE speed (like 115.2k) then flow from your modem to your PC may work OK but uploading flow in the other direction will not all get thru due to the telephone line bottleneck. This will result in many errors and the resending of packets. It may thus take far too long to send a file. In some cases, files don’t make it thru at all

For the downloading case: If you’re downloading long uncompressed files or web pages (and your modem uses data compression) or if you’ve set a low DTE speed, then downloading may also be broken due to no flow control.

For Dial−in I Keep Getting "line NNN of inittab invalid"

Make sure you are using the correct syntax for your version of init. The different init’s that are out there use different syntax in the /etc/inittab file. Make sure you are using the correct syntax for your version of getty.

I Keep Getting: “Id "S3" respawning too fast: disabled for 5 minutes”

Id "S3" is just an example. In this case look on the line which starts with "S3" in /etc/inittab and calls getty. This line causes the problem. Make sure the syntax for this line is correct and that the device (ttyS3) exists and can be found. If the modem has negated CD and getty opens the port, you’ll get this error message since negated CD will kill getty. Then getty will respawn only to be killed again, etc. Thus it respawns over and over (too fast). It seems that if the cable to the modem is disconnected or you have the wrong serial port, it’s just like CD is negated. All this can occur when your modem is chatting with getty. Make sure your modem is configured correctly. Look at AT commands E and Q.

If you use uugetty, verify that your /etc/gettydefs syntax is correct by doing the following:

```
linux# getty −c /etc/gettydefs
```

This can also happen when the uugetty initialization is failing. See section uugetty Still Doesn’t Work.

My Modem is Hosed after Someone Hangs Up, or uugetty doesn't respawn

This can happen when your modem doesn't reset when DTR is dropped. Greg Hankins saw his RD and SD LEDs go crazy when this happened. You need to have your modem reset. Most Hayes compatible modems do this with &D3, but for USR Courier, SupraFax, and other modems, you must set &D2 (and S13=1 for USR Courier). Check your modem manual (if you have one).

uugetty Still Doesn't Work

There is a DEBUG option that comes with getty_ps. Edit your config file /etc/conf.{uu}getty.ttySN and add DEBUG=NNN. Where NNN is one of the following combination of numbers according to what you are trying to debug:

```
D_OPT   001         option settings
D_DEF   002         defaults file processing
D_UTMP  004         utmp/wtmp processing
D_INIT  010         line initialization (INIT)
D_GTAB  020         gettytab file processing
D_RUN   040         other runtime diagnostics
D_RB    100         ringback debugging
D_LOCK  200         uugetty lockfile processing
D_SCH   400         schedule processing
D_ALL   777         everything
```

Setting DEBUG=010 is a good place to start.
If you are running syslogd, debugging info will appear in your log files. If you aren't running syslogd info will appear in /tmp/getty:ttySN for debugging getty and /tmp/uugetty:ttySN for uugetty, and in /var/adm/getty.log. Look at the debugging info and see what is going on. Most likely, you will need to tune some of the parameters in your config file, and reconfigure your modem.
You could also try mgetty. Some people have better luck with it.

My Serial Port is Physically There but Can't be Found

If a physical device (such as a modem) doesn't work at all it may mean that the device is not at the I/O address that setserial thinks it's at. It could also mean (for a PnP card) that is doesn't yet have an address. Thus it can't be found.
Check the BIOS menus and BIOS messages. For the PCI bus use lspci or scanpci. If it's an ISA bus PnP serial port, try "pnpdump −−dumpregs" and/or see Plug−and−Play−HOWTO. Using "scanport" will scan all ISA bus ports and may discover an unknown port that could be a serial port (but it doesn't probe the port). It could hang your PC. You may try probing with setserial. See Probing. If nothing seems to get thru the port it may be accessible but have a bad interrupt. See Extremely Slow: Text appears on the screen slowly after long delays. Use setserial −g to see what the serial driver thinks and check for IRQ and I0 address conflicts. Even if you see no conflicts the driver may have incorrect information (view it by "setserial" and conflicts may still exist.
If two ports have the same IO address then probing it will erroneously indicate only one port. Plug−and−play detection will find both ports so this should only be a problem if at least one port is not plug−and−play. All sorts of errors may be reported/observed for devices illegally "sharing" a port but the fact that there are two devices on the same a port doesn't seem to get detected (except hopefully by you). In the above case, if the IRQs are different then probing for IRQs with setserial might "detect" this situation by failing to detect any IRQ. See Probing.

Extremely Slow: Text appears on the screen slowly after long delays

It's likely mis−set/conflicting interrupts. Here are some of the symptoms which will happen the first time you try to use a modem, terminal, or serial printer. In some cases you type something but nothing appears on the screen until many seconds later. Only the last character typed may show up. It may be just an invisible <return> character so all you notice is that the cursor jumps down one line.

In other cases where a lot of data should appear on the screen, only a batch of about 16 characters appear. Then there is a long wait of many seconds for the next batch of characters. You might also get "input overrun" error messages (or find them in logs).
For more details on the symptoms and why this happens see the Serial−HOWTO section: "Interrupt Problem Details".
If it involves Plug−and−Play devices, see also Plug−and−Play−HOWTO.
As a quick check to see if it really is an interrupt problem, set the IRQ to 0 with "setserial". This will tell the driver to use polling instead of interrupts. If this seems to fix the "slow" problem then you had an interrupt problem. You should still try to solve the problem since polling uses excessive computer resources.
Checking to find the interrupt conflict may not be easy since Linux supposedly doesn't permit any interrupt conflicts and will send you a /dev/ttyS?: Device or resource busy error message if it thinks you are attempting to create a conflict. But a real conflict can be created if "setserial" has told the kernel incorrect info. The kernel has been lied to and thus doesn't think there is any conflict. Thus using "setserial" will not reveal the conflict (nor will looking at /proc/interrupts which bases
its info on "setserial"). You still need to know what "setserial" thinks so that you can pinpoint where it's wrong and change it when you determine what's really set in the hardware.
What you need to do is to check how the hardware is set by checking jumpers or using PnP software to check how the hardware is actually set. For PnP run either "pnpdump −−dumpregs" (if ISA bus) or run "lspci" (if PCI bus). Compare this to how Linux (e.g. "setserial") thinks the hardware is set.

Somewhat Slow: I expected it to be a few times faster

One reason may be that whatever is on the serial port (such as a modem, terminal, printer) doesn't work as fast as you thought it did. A 56k Modem seldom works at 56k and the Internet often has congestion and bottlenecks that slow things down. If the modem on the other end does not have a digital connection to the phone line (and uses a special "digital modem" not sold in most computer stores), then speeds above 33.6k are not possible.
Another possible reason is that you have an obsolete serial port: UART 8250, 16450 or early 16550 (or the serial driver thinks you do). See "What are UARTS" in the Serial−HOWTO.
Use "setserial −g /dev/ttyS*". If it shows anything less than a 16550A, this may be your problem. If you think that "setserial" has it wrong check it out. See What is Setserial for more info. If you really do have an obsolete serial port, lying about it to setserial will only make things worse.

The Startup Screen Show Wrong IRQs for the Serial Ports.

Linux does not do any IRQ detection on startup. When the serial module loads it only does serial device detection. Thus, disregard what it says about the IRQ, because it's just assuming the standard IRQs. This is done, because IRQ detection is unreliable, and can be fooled. But if and when setserial runs from a start−up script, it changes the IRQ's and displays the new (and hopefully correct) state on on the startup screen. If the wrong IRQ is not corrected by a later display on the screen, then you've got a problem.
So, even though I have my ttyS2 set at IRQ 5, I still see
 ttyS02 at 0x03e8 (irq = 4) is a 16550A
at first when Linux boots. (Older kernels may show "ttyS02" as "tty02" which is the same as ttyS2). You may need to use setserial to tell Linux the IRQ you are using.

"Cannot open /dev/ttyS?: Permission denied"

Check the file permissions on this port with "ls −l /dev/ttyS?"_ If you own the ttyS? then you need read and write permissions: crw with the c (Character device) in col. 1. It you don't own it then it should show rw− in cols. 8 & 9 which means that everyone has read and write permission on it. Use "chmod" to change permissions. There are more complicated ways to get access like belonging to a "group" that has group permission.

"Operation not supported by device" for ttyS?

This means that an operation requested by setserial, stty, etc. couldn't be done because the kernel doesn't support doing it. Formerly this was often due to the "serial" module not being loaded. But with the advent of PnP, it may likely mean that there is no modem (or other serial device) at the address where the driver (and setserial) thinks it is. If there is no modem there, commands (for operations) sent to that address obviously don't get done. See What is set in my serial port hardware?
If the "serial" module wasn't loaded but "lsmod" shows you it's now loaded it might be the case that it's loaded now but wasn't loaded when you got the error message. In many cases the module will automatically loaded when needed (if it can be found). To force loading of the "serial" module it may be listed in the file: /etc/modules.conf or /etc/modules. The actual module should reside in: /lib/modules/.../misc/serial.o.

"Cannot create lockfile. Sorry"

When a port is "opened" by a program a lockfile is created in /var/lock/. Wrong permissions for the lock directory will not allow a lockfile to be created there. Use "ls −ld /var/lock" to see if the permissions are OK: usually rwx for everyone (repeated 3 times). If it's wrong, use "chmod" to fix it. Of course, if there is no "lock" directory no lockfile can be created there. For more info on lockfiles see the Serial−HOWTO subsection:"What Are Lock Files".

"Device /dev/ttyS? is locked."

This means that someone else (or some other process) is supposedly using the serial port. There are various ways to try to find out what process is "using" it. One way is to look at the contents of the lockfile (/var/lock/LCK...). It should be the process id. If the process id is say 100 type "ps 100" to find out what it is. Then if the process is no longer needed, it may be gracefully killed by "kill 100". If it refuses to be killed use "kill −9 100" to force it to be killed, but then the lockfile will not be removed and you'll need to delete it manually. Of course if there is no such process as 100 then you may just remove the lockfile but in most cases the lockfile should have been automatically removed if it contained a stale process id (such as 100).

"/dev/tty? Device or resource busy"

This means that the device you are trying to access (or use) is supposedly busy (in use) or that a resource it needs (such as an IRQ) is supposedly being used by another device (the resource is "busy"). This message is easy to understand if it only means that the device is busy (in use). But it often means that a resource is in use. What makes it even more confusing is that in some cases neither the device not the resources that it needs are actually "busy".
The ‘‘resource busy’’ part often means (example for ttyS2) ‘‘You can't use ttyS2 since another device is using ttyS2's interrupt.’’ The potential interrupt conflict is inferred from what "setserial" thinks. A more accurate error message would be ‘‘Can't use ttyS2 since the setserial data (and kernel data) indicates that another device is using ttyS2's interrupt’’. If two devices use the same IRQ and you start up only one of the devices, everything is OK because there is no conflict yet. But when you next try to start the second device (without quitting the first device) you get a "... busy" error message. This is because the kernel only keeps track of what RQs are actually in use and actual conflicts don't happen unless the devices are in use (open). The situation for I/O address (such as 0x3f8) conflict is similar.
This error is sometimes due to having two serial drivers: one a module and the other compiled into the kernel. Both drivers try to grab the same resources and one driver finds them "busy".
There are two possible cases when you see this message:

1. There may be a real resource conflict that is being avoided.
2. Setserial has it wrong and the only reason ttyS2 can't be used is that setserial erroneously predicts a conflict.

What you need to do is to find the interrupt setserial thinks ttyS2 is using. Look at /proc/tty/driver/serial (if you have it). You should also be able to find it with the "setserial" command for ttyS2. But due to a bug (reported by me in Nov. 2000) you get the same "... busy" error message when you try this with "setserial".
To try to resolve this problem reboot or: exit or gracefully kill all likely conflicting processes. If you reboot: 1. Watch the boot−time messages for the serial ports. 2. Hope that the file that runs "setserial" at boot−time doesn't (by itself) create the same conflict again.
If you think you know what IRQ say ttyS2 is using then you may look at /proc/interrupts to find what else (besides another serial port) is currently using this IRQ. You might also want to double check that any suspicious IRQs shown here (and by "setserial") are correct (the same as set in the hardware). A way to test whether or not it's a potential interrupt conflict is to set the IRQ to 0 (polling) using "setserial". Then if the busy message goes away, it was likely a potential interrupt conflcit. It's not a good idea to leave it permanently set at 0 since it will make the CPU work too hard.

"Input/output error" from setserial or stty

You may have typed "ttys" instead of "ttyS". You will see this error message if you try to use the setserial command for any device that is not a serial port. It also may mean that the serial port is in use (busy or opened) and thus the attempt to get/set parameters by setserial or stty failed. It could also mean that there isn't any serial port at the IO address that setserial thinks your port is at.

Overrun errors on serial port

This is an overrun of the hardware FIFO buffer and you can't increase its size. See "Higher Serial Thruput" in the Serial−HOWTO.

Modem doesn't pick up incoming calls

This paragraph is for the case where a modem is used for both dial−in and dial−out. If the modem generates a DCD (=CD) signal, some programs (but not mgetty) will think that the modem is busy. This will cause a problem when you are trying to dial out with a modem and the modem's DCD or DTR are not
implemented correctly. The modem should assert DCD only hen there is an actual connection (ie someone has dialed in), not when getty is watching the
port. Check to make sure that your modem is configured to only assert DCD when there is a connection (&C1). DTR should be on (asserted) by the
communications program whenever something is using, or watching the line, like getty, kermit, or some other comm program.

Troubleshooting Tools

These are some of the programs you might want to use in troubleshooting:

- "lsof /dev/ttyS*" will list serial ports which are open.
- "setserial" shows and sets the low−level hardware configuration of a port (what the driver thinks it is). See What is Setserial
- "stty" shows and sets the configuration of a port (except for that handled by "setserial"). See the Serial−HOWTO section: "Stty".
- "modemstat" or "statserial" will show the current state of various modem signal lines (such as DTR, CTS, etc.)
- "irqtune" will give serial port interrupts higher priority to improve performance.
- "hdparm" for hard−disk tuning may help some more.
- "lspci" shows the actual IRQs, etc. of hardware on the PCI bus.
- "pnpdump −−dumpregs" shows the actual IRQs, etc. of hardware for PnP devices on the ISA bus.
- Some "files" in the /proc tree (such as ioports, interrupts, and tty/driver/serial).

Version showing changes of each author, pages 1–2

1

Troubleshooting

My Modem is Physically There but Can't be Found

2

The error messages could be something like "No modem detected", "Modem not responding", or (strange) "You are already online" (from Minicom). If you have installed an internal modem (serial port is builtin) or are using an external one and don't know what serial port it's connected to then the problem is to find the serial port. See My Serial Port is Physically There but Can't be Found. This section is about finding out which serial port has the modem on it.

The error messages could be something like "No modem detected", "Modem not responding", or (strange) "You are already online" (from Minicom). If you have installed an internal modem (serial port is builtin) or are using an external one and don't know what serial port it's connected to then the problem is to find the serial port. See *My Serial Port is Physically There but Can't be Found*. This section is about finding out which serial port has the modem on it.

3

There's a program that looks for modems on commonly used serial ports called "wvdialconf". Just type "wvdialconf <a–new–file–name>". It will create the new file as a configuration file but you don't need this file unless you are going to use "wvdial" for dialing. See What is wvdialconf ? Unfortunately, if your modem is in "online data" mode, wvdialconf will report "No modem detected" See No response to AT

There's a program that looks for modems on commonly used serial ports called "wvdialconf". Just type "wvdialconf <a–new–file–name>". It will create the new file as a configuration file but you don't need this file unless you are going to use "wvdial" for dialing. See What is wvdialconf? Unfortunately, if your modem is in "online data" mode, wvdialconf will report "No modem detected" See No response to AT

3

Your problem could be due to a winmodem (or the like) which usually can't be used with Linux. See Software–based Modems (winmodems). The "setserial program may be used to detect serial ports but will not detect modems on them. Thus "wvdialconf" is best to try first.

Your problem could be due to a winmodem (or the like) which usually can't be used with Linux. ~~See Software–based Modems (winmodems).~~ The "setserial program may be used to detect serial ports but will not detect modems on them. Thus "wvdialconf" is best to try first.

4

Another way try to find out if there's a modem on a port is to start "minicom" on the port (after first setting up minicom for the correct serial port – –you will need to save the setup and then exit minicom and start it again). Then type "AT" and you should see OK (or 0 if it's set for "digit result codes"). The results may be:

Another way try to find out if there's a modem on a port is to start "minicom" on the port (after first setting up minicom for the correct serial port – you will need to save the setup and then exit minicom and start it again). Then type "AT" and you should see OK (or 0 if it's set for "digit result codes"). The results may be:

5

- No response. See No response to AT
- It takes many seconds to get an expected truncated response (including only the cursor moving down one line). See Extremely Slow: Text appears on the screen slowly after long delays
- * Some strange characters appear but they are not in response to AT. This likely means that your modem is still connected to something at the other end of the phone line which is sending some cryptic packets or the like.
- No response. See *No response to AT*
- It takes many seconds to get an expected truncated response (including only the cursor moving down one line). See Extremely Slow: Text appears on the screen slowly after long delays
- ~~*~~Some strange characters appear but they are not in response to AT. This likely means that your modem is still connected to something at the other end of the phone line which is sending some cryptic packets or the like.

6

No response to AT

No response to AT

The modem should send you "OK" in response to your "AT" which you type to the modem (using minicom or the like). If you don't see "OK" (and in most cases don't even see the "AT" you typed either) then the modem is not responding (often because what you type doesn't even get to the modem).

7

A common cause is that there is no modem on the serial port you are typing to. For the case of an internal modem, that serial port likely doesn't exist either. That's because the PnP modem card (which has a built–in serial port) has either not been configured (by isapnp or the like) or has been configured incorrectly. See My Serial Port is Physically There but Can't be Found.

A common cause is that there is no modem on the serial port you are typing to. For the case of an internal modem, that serial port likely doesn't exist either. That's because the PnP modem card (which has a built–in serial port) has either not been configured ~~(by isapnp or the like)~~ or has been configured incorrectly. See My Serial Port is Physically There but Can't be Found.

If what you type is really getting thru to a modem, then the lack of response could be due to the modem being in "online data" mode where it can't accept any AT commands. You may have been using the modem and then abruptly disconnected (such as killing the process with signal 9). In that case your modem did not get reset to "command mode" where it can interact to AT commands. Thus the message from minicom "You are already online. Hangup first." Well, you are sort of online but you are may not be connected to anything over the phone line. Wvdial will report "modem not responding" for the same situation.

To fix this as a last resort you could reboot the computer. Another way to try to fix this is to send +++ to the modem to tell it to escape back to "command mode" from "online data mode". On both sides of the +++ sequence there must be about 1 second of delay (nothing sent during "guard time"). This may not work if another process is using the modem since the +++ sequence could wind up with other characters inserted in between them or after the +++ (during the guard time). Ironically, even if the modem line is idle, typing an unexpected +++ is likely to set off an exchange of control packets (that you never see) that will violate the required guard time so that the +++ doesn't do what you wanted. +++ is usually in the string that is named "hangup string" so if you command minicom (or the like) to hangup it might work. Another way to do this is to just exit minicom and then run minicom again.

8

My Serial Port is Physically There but Can't be Found

If a physical device (such as a modem) doesn't work at all it may mean that the device is not at the I/O address that setserial thinks it's at. It could also mean (for a PnP card) that is doesn't yet have an address. Thus it can't be found.

Check the BIOS menus and BIOS messages. For the PCI bus use lspci or scanpci. If it's an ISA bus PnP serial port, try "pnpdump ––dumpregs" and/or see Plug–and–Play–HOWTO. Using "scanport" will scan all ISA bus ports and may discover an unknown port that could be a serial port (but it doesn't probe the port). It could hang your PC. You may try probing with setserial. See Probing. If nothing seems to get thru to the port it may be accessible but have a bad interrupt. See Extremely Slow: Text appears on the screen slowly after long delays. Use setserial –g to see what the serial driver thinks and check for IRQ and I0 address conflicts. Even if you see no conflicts the driver may have

Version showing changes of each author, pages 3–4

incorrect information (view it by "setserial" and conflicts may still exist.

If two ports have the same IO address then probing it will erroneously indicate only one port. Plug–and–play detection will find both ports so this should only be a problem if at least one port is not plug–and–play. All sorts of errors may be reported/observed for devices illegally "sharing" a port but the fact that there are two devices on the same a port doesn't seem to get detected (except hopefully by you). In the above case, if the IRQs are different then probing for IRQs with setserial might "detect" this situation by failing to detect any IRQ. See Probing.

"Modem is busy"

What this means depends on what program sent it. The modem could actually be in use (busy). Another cause reported for the SuSE distribution is that there may be two serial drivers present instead of one. One driver was built into the kernel and the second was a module.

In kppp, this message is sent when an attempt to get/set the serial port parameters fails. These parameters are the one you see if you give the "stty –a" command. It's similar to the "Input/output error" one may get when trying to use "stty –F /dev/ttySx". Although the port is already opened when you see this message, I think it's possible to open a non–existent device (or one with the wrong IRQ or IO address). Now getting certain port parameters (such as the speed) means communicating with the serial port hardware. So this error message may mean that there is no serial port there (or that the modem's serial port has an incorrect (or no) IRQ or I0 address. It could be a wrong ttySx number. If /dev/modem is used it should be linked to the correct ttySx. In these cases the error message should have said: "Modem can't be found" which really means that it's serial port (often built into the modem) can't be found.

9

In kppp, this message is sent when an attempt to get/set the serial port parameters fails. These parameters are the one you see if you give the `stty -a` command. It's similar to the "Input/output error" one may get when trying to use `stty -F /dev/ttySx`. Although the port is already opened when you see this message, I think it's possible to open a non–existent device (or one with the wrong IRQ or IO address). Now getting certain port parameters (such as the speed) means communicating with the serial port hardware. So this error message may mean that there is no serial port there (or that the modem's serial port has an incorrect (or no) IRQ or I0 address. It could be a wrong ttySx number. If /dev/modem is used it should be linked to the correct ttySx. In these cases the error message should have said: "Modem can't be found" which really means that it's serial port (often built into the modem) can't be found.

I can't get near 56k on my 56k modem

There must be very low noise on the line for it to work at even close to 56k. Some phone lines are so bad that the speeds obtainable are much slower than 56k (like 28.8k or even slower). Sometimes extension phones connected to the same line can cause problems. To test this you might connect your modem directly at the point where the telephone line enters the building with the feeds for everything else on that line disconnected (if others can tolerate such a test).

Uploading (downloading) files is broken/slow

Flow control (both at your PC and/or modem–to–modem) may not be enabled. For the uploading case: If you have set a high DTE speed (like 115.2k) then flow from your modem to your PC may work OK but uploading flow in the other direction will not all get thru due to the telephone line bottleneck. This will result in many errors and the resending of packets. It may thus take far too long to send a file. In some cases, files don't make it thru at all

For the downloading case: If you're downloading long uncompressed files or web pages (and your modem uses data compression) or if you've set a low DTE speed, then downloading may also be broken due to no flow control.

10

For Dial–in I Keep Getting "line NNN of inittab invalid"

Make sure you are using the correct syntax for your version of init. The different init's that are out there use different syntax in the /etc/inittab file. Make sure you are using the correct syntax for your version of getty.

~~*For Dial–in I Keep Getting "line NNN of inittab invalid"*~~

~~Make sure you are using the correct syntax for your version of init. The different init's that are out there use different syntax in the /etc/inittab file. Make sure you are using the correct syntax for your version of getty.~~

For Dial–in I Keep Getting "line NNN of inittab invalid"

Make sure you are using the correct syntax for your version of init. The different init's that are out there use different syntax in the /etc/inittab file. Make sure you are using the correct syntax for your version of getty (see man getty).

I Keep Getting: "Id "S3" respawning too fast: disabled for 5 minutes"

Id "S3" is just an example. In this case look on the line which starts with "S3" in /etc/inittab and calls getty. This line causes the problem. Make sure the syntax for this line is correct and that the device (ttyS3) exists and can be found. If the modem has negated CD and getty opens the port, you'll get this error message since negated CD will kill getty. Then getty will respawn only to be killed again, etc. Thus it respawns over and over (too fast). It seems that if the cable to the modem is disconnected or you have the wrong serial port, it's just like CD is negated. All this can occur when your modem is chatting with getty. Make sure your modem is configured correctly. Look at AT commands E and Q.

If you use uugetty, verify that your /etc/gettydefs syntax is correct by doing the following:

11

linux# getty –c /etc/gettydefs

```
linux# getty -c /etc/gettydefs
```

This can also happen when the uugetty initialization is failing. See section uugetty Still Doesn't Work.

My Modem is Hosed after Someone Hangs Up, or uugetty doesn't respawn

This can happen when your modem doesn't reset when DTR is dropped. Greg Hankins saw his RD and SD LEDs go crazy when this happened. You need to have your modem reset. Most Hayes compatible modems do this with &D3, but for USR Courier, SupraFax, and other modems, you must set &D2 (and S13=1 for USR Courier). Check your modem manual (if you have one).

uugetty Still Doesn't Work

There is a DEBUG option that comes with getty_ps. Edit your config file /etc/conf.{uu}getty.ttySN and add DEBUG=NNN. Where NNN is one of the following combination of numbers according to what you are trying to debug:

```
D_OPT    001          option settings
D_DEF    002          defaults file processing
D_UTMP   004          utmp/wtmp processing
D_INIT   010          line initialization (INIT)
D_GTAB   020          gettytab file processing
```

Version showing changes of each author, pages 5–6

12

```
D_RUN   040           other runtime diagnostics
D_RB    100           ringback debugging
D_LOCK  200           uugetty lockfile processing
D_SCH   400           schedule processing
D_ALL   777           everything
```

```
D_OPT   001           option settings
D_DEF   002           defaults file processing
D_UTMP  004           utmp/wtmp processing
D_INIT  010           line initialization (INIT)
D_GTAB  020           gettytab file processing
D_RUN   040           other runtime diagnostics
D_RB    100           ringback debugging
D_LOCK  200           uugetty lockfile processing
D_SCH   400           schedule processing
D_ALL   777           everything
```

Setting DEBUG=010 is a good place to start.

13

If you are running syslogd, debugging info will appear in your log files. If you aren't running syslogd info will appear in /tmp/getty:ttySN for debugging getty and /tmp/uugetty:ttySN for uugetty, and in /var/adm/getty.log. Look at the debugging info and see what is going on. Most likely, you will need to tune some of the parameters in your config file, and reconfigure your modem.

If you are running syslogd, debugging info will appear in your log files. If you aren't running syslogd info will appear in `/tmp/getty:ttySN` for debugging getty and `/tmp/uugetty:ttySN` for uugetty, and in `/var/adm/getty.log`. Look at the debugging info and see what is going on. Most likely, you will need to tune some of the parameters in your config file, and reconfigure your modem.

You could also try mgetty. Some people have better luck with it.

14

My Serial Port is Physically There but Can't be Found

If a physical device (such as a modem) doesn't work at all it may mean that the device is not at the I/O address that setserial thinks it's at. It could also mean (for a PnP card) that is doesn't yet have an address. Thus it can't be found.

Check the BIOS menus and BIOS messages. For the PCI bus use lspci or scanpci. If it's an ISA bus PnP serial port, try "pnpdump −−dumpregs" and/or see Plug−and−Play−HOWTO. Using "scanport" will scan all ISA bus ports and may discover an unknown port that could be a serial port (but it doesn't probe the port). It could hang your PC. You may try probing with setserial. See Probing. If nothing seems to get thru the port it may be accessible but have a bad interrupt. See Extremely Slow: Text appears on the screen slowly after long delays. Use setserial −g to see what the serial driver thinks and check for IRQ and I0 address conflicts. Even if you see no conflicts the driver may have incorrect information (view it by "setserial" and conflicts may still exist.

If two ports have the same IO address then probing it will erroneously indicate only one port. Plug−and−play detection will find both ports so this should only be a problem if at least one port is not plug−and−play. All sorts of errors may be reported/observed for devices illegally "sharing" a port but the fact that there are two devices on the same a port doesn't seem to get detected (except hopefully by you). In the above case, if the IRQs are different then probing for IRQs with setserial might "detect" this situation by failing to detect any IRQ. See Probing.

Extremely Slow: Text appears on the screen slowly after long delays

It's likely mis−set/conflicting interrupts. Here are some of the symptoms which will happen the first time you try to use a modem, terminal, or serial printer. In some cases you type something but nothing appears on the screen until many seconds later. Only the last character typed may show up. It may be just an invisible <return> character so all you notice is that the cursor jumps down one line. In other cases where a lot of data should appear on the screen, only a batch of about 16 characters appear. Then there is a long wait of many seconds for the next batch of characters. You might also get "input overrun" error messages (or find them in logs).

15

For more details on the symptoms and why this happens see the Serial−HOWTO section: "Interrupt Problem Details".

~~For more details on the symptoms and why this happens see the Serial−HOWTO section: "Interrupt Problem Details".~~

If it involves Plug−and−Play devices, see also Plug−and−Play−HOWTO.

As a quick check to see if it really is an interrupt problem, set the IRQ to 0 with "setserial". This will tell the driver to use polling instead of interrupts. If this seems to fix the "slow" problem then you had an interrupt problem. You should still try to solve the problem since polling uses excessive computer resources.

Checking to find the interrupt conflict may not be easy since Linux supposedly doesn't permit any interrupt conflicts and will send you a /dev/ttyS?: Device or resource busy error message if it thinks you are attempting to create a conflict. But a real conflict can be created if "setserial" has told the kernel incorrect info. The kernel has been lied to and thus doesn't think there is any conflict. Thus using "setserial" will not reveal the conflict (nor will looking at /proc/interrupts which bases

its info on "setserial"). You still need to know what "setserial" thinks so that you can pinpoint where it's wrong and change it when you determine what's really set in the hardware.

What you need to do is to check how the hardware is set by checking jumpers or using PnP software to check how the hardware is actually set. For PnP run either "pnpdump −−dumpregs" (if ISA bus) or run "lspci" (if PCI bus). Compare this to how Linux (e.g. "setserial") thinks the hardware is set.

Somewhat Slow: I expected it to be a few times faster

One reason may be that whatever is on the serial port (such as a modem, terminal, printer) doesn't work as fast as you thought it did. A 56k Modem seldom works at 56k and the Internet often has congestion and bottlenecks that slow things down. If the modem on the other end does not have a digital connection to the phone line (and uses a special "digital modem" not sold in most computer stores), then speeds above 33.6k are not possible.

16

Another possible reason is that you have an obsolete serial port: UART 8250, 16450 or early 16550 (or the serial driver thinks you do). See "What are UARTS" in the Serial−HOWTO.

~~Another possible reason is that you have an obsolete serial port: UART 8250, 16450 or early 16550 (or the serial driver thinks you do). See "What are UARTS" in the Serial−HOWTO.~~

Use "setserial −g /dev/ttyS*". If it shows anything less than a 16550A, this may be your problem. If you think that "setserial" has it wrong check it out. See What is Setserial for more info. If you really do have an obsolete serial port, lying about it to setserial will only make things worse.

The Startup Screen Show Wrong IRQs for the Serial Ports.

Linux does not do any IRQ detection on startup. When the serial module loads it only does serial device detection. Thus, disregard what it says about the IRQ, because it's just assuming the standard IRQs. This is done, because IRQ detection is unreliable, and can be fooled. But if and when setserial runs from a start−up script, it changes the IRQ's and displays the new (and hopefully correct) state

Version showing changes of each author, pages 7–8

on on the startup screen. If the wrong IRQ is not corrected by a later display on the screen, then you've got a problem.

So, even though I have my ttyS2 set at IRQ 5, I still see

ttyS02 at 0x03e8 (irq = 4) is a 16550A

at first when Linux boots. (Older kernels may show "ttyS02" as "tty02" which is the same as ttyS2). You may need to use setserial to tell Linux the IRQ you are using.

17

"Cannot create lockfile. Sorry"

When a port is "opened" by a program a lockfile is created in /var/lock/. Wrong permissions for the lock directory will not allow a lockfile to be created there. Use "ls −ld /var/lock" to see if the permissions are OK: usually rwx for everyone (repeated 3 times). If it's wrong, use "chmod" to fix it. Of course, if there is no "lock" directory no lockfile can be created there. For more info on lockfiles see the Serial−HOWTO subsection:"What Are Lock Files".

"Cannot open /dev/ttyS?: Permission denied"

Check the file permissions on this port with "ls −l /dev/ttyS?"_ If you own the ttyS? then you need read and write permissions: crw with the c (Character device) in col. 1. It you don't own it then it should show rw− in cols. 8 & 9 which means that everyone has read and write permission on it. Use "chmod" to change permissions. There are more complicated ways to get access like belonging to a "group" that has group permission.

"Operation not supported by device" for ttyS?

This means that an operation requested by setserial, stty, etc. couldn't be done because the kernel doesn't support doing it. Formerly this was often due to the "serial" module not being loaded. But with the advent of PnP, it may likely mean that there is no modem (or other serial device) at the address where the driver (and setserial) thinks it is. If there is no modem there, commands (for operations) sent to that address obviously don't get done. See What is set in my serial port hardware?

If the "serial" module wasn't loaded but "lsmod" shows you it's now loaded it might be the case that it's loaded now but wasn't loaded when you got the error message. In many cases the module will automatically loaded when needed (if it can be found). To force loading of the "serial" module it may be listed in the file: /etc/modules.conf or /etc/modules. The actual module should reside in: /lib/modules/.../misc/serial.o.

18

"Cannot create lockfile. Sorry"

When a port is "opened" by a program a lockfile is created in /var/lock/. Wrong permissions for the lock directory will not allow a lockfile to be created there. Use "ls −ld /var/lock" to see if the permissions are OK: usually rwx for everyone (repeated 3 times). If it's wrong, use "chmod" to fix it. Of course, if there is no "lock" directory no lockfile can be created there. For more info on lockfiles see the Serial−HOWTO subsection:"What Are Lock Files".

"Device /dev/ttyS? is locked."

This means that someone else (or some other process) is supposedly using the serial port. There are various ways to try to find out what process is "using" it. One way is to look at the contents of the lockfile (/var/lock/LCK...). It should be the process id. If the process id is say 100 type "ps 100" to find out what it is. Then if the process is no longer needed, it may be gracefully killed by "kill 100". If it refuses to be killed use "kill −9 100" to force it to be killed, but then the lockfile will not be removed and you'll need to delete it manually. Of course if there is no such process as 100 then you may just remove the lockfile but in most cases the lockfile should have been automatically removed if it contained a stale process id (such as 100).

"/dev/tty? Device or resource busy"

This means that the device you are trying to access (or use) is supposedly busy (in use) or that a resource it needs (such as an IRQ) is supposedly being used by another device (the resource is "busy"). This message is easy to understand if it only means that the device is busy (in use). But it often means that a resource is in use. What makes it even more confusing is that in some cases neither the device not the resources that it needs are actually "busy".

The ''resource busy'' part often means (example for ttyS2) ''You can't use ttyS2 since another device is using ttyS2's interrupt.'' The potential interrupt conflict is inferred from what "setserial" thinks. A more accurate error message would be ''Can't use ttyS2 since the setserial data (and kernel data) indicates that another device is using ttyS2's interrupt''. If two devices use the same IRQ and you start up only one of the devices, everything is OK because there is no conflict yet. But when you next try to start the second device (without quitting the first device) you get a "... busy" error message. This is because the kernel only keeps track of what RQs are actually in use and actual conflicts don't happen unless the devices are in use (open). The situation for I/O address (such as 0x3f8) conflict is similar.

This error is sometimes due to having two serial drivers: one a module and the other compiled into the kernel. Both drivers try to grab the same resources and one driver finds them "busy".

There are two possible cases when you see this message:

1. There may be a real resource conflict that is being avoided.
2. Setserial has it wrong and the only reason ttyS2 can't be used is that

   setserial erroneously predicts a conflict.

What you need to do is to find the interrupt setserial thinks ttyS2 is using. Look at /proc/tty/driver/serial (if you have it). You should also be able to find it with the "setserial" command for ttyS2. But due to a bug (reported by me in Nov. 2000) you get the same "... busy" error message when you try this with "setserial".

To try to resolve this problem reboot or: exit or gracefully kill all likely conflicting processes. If you reboot: 1. Watch the boot−time messages for the serial ports. 2. Hope that the file that runs "setserial" at boot−time doesn't (by itself) create the same conflict again.

If you think you know what IRQ say ttyS2 is using then you may look at /proc/interrupts to find what else (besides another serial port) is currently using this IRQ. You might also want to double check that any suspicious IRQs shown here (and by "setserial") are correct (the same as set in the hardware). A way to test whether or not it's a potential interrupt conflict is to set the IRQ to 0 (polling) using "setserial". Then if the busy message goes away, it was likely a potential interrupt conflcit. It's not a good idea to leave it permanently set at 0 since it will make the CPU work too hard.

19

If you think you know what IRQ say ttyS2 is using then you may look at /proc/interrupts to find what else (besides another serial port) is currently using this IRQ. You might also want to double check that any suspicious IRQs shown here (and by "setserial") are correct (the same as set in the hardware). A way to test whether or not it's a potential interrupt conflict is to set the IRQ to 0 (polling) using "setserial". Then if the busy message goes away, it was likely a potential interrupt conflict. It's not a good idea to leave it permanently set at 0 since it will make the CPU work too hard.

Version showing changes of each author, pages 9–10

20

"Cannot create lockfile. Sorry"

When a port is "opened" by a program a lockfile is created in /var/lock/. Wrong permissions for the lock directory will not allow a lockfile to be created there. Use "ls −ld /var/lock" to see if the permissions are OK: usually rwx for everyone (repeated 3 times). If it's wrong, use "chmod" to fix it. Of course, if there is no "lock" directory no lockfile can be created there. For more info on lockfiles see the Serial−HOWTO subsection:"What Are Lock Files".

"Input/output error" from setserial or stty

You may have typed "ttys" instead of "ttyS". You will see this error message if you try to use the setserial command for any device that is not a serial port. It also may mean that the serial port is in use (busy or opened) and thus the attempt to get/set parameters by setserial or stty failed. It could also mean that there isn't any serial port at the IO address that setserial thinks your port is at.

Overrun errors on serial port

This is an overrun of the hardware FIFO buffer and you can't increase its size. See "Higher Serial Thruput" in the Serial−HOWTO.

Modem doesn't pick up incoming calls

This paragraph is for the case where a modem is used for both dial−in and dial−out. If the modem generates a DCD (=CD) signal, some programs (but not mgetty) will think that the modem is busy. This will cause a problem when you are trying to dial out with a modem and the modem's DCD or DTR are not

implemented correctly. The modem should assert DCD only hen there is an actual connection (ie someone has dialed in), not when getty is watching the

port. Check to make sure that your modem is configured to only assert DCD when there is a connection (&C1). DTR should be on (asserted) by the

communications program whenever something is using, or watching the line, like getty, kermit, or some other comm program.

Troubleshooting Tools

These are some of the programs you might want to use in troubleshooting:

- "lsof /dev/ttyS*" will list serial ports which are open.
- "setserial" shows and sets the low−level hardware configuration of a port (what the driver thinks it is). See What is Setserial
- "stty" shows and sets the configuration of a port (except for that handled by "setserial"). See the Serial−HOWTO section: "Stty".
- "modemstat" or "statserial" will show the current state of various modem signal lines (such as DTR, CTS, etc.)
- "irqtune" will give serial port interrupts higher priority to improve performance.
- "hdparm" for hard−disk tuning may help some more.
- "lspci" shows the actual IRQs, etc. of hardware on the PCI bus.
- "pnpdump −−dumpregs" shows the actual IRQs, etc. of hardware for PnP devices on the ISA bus.
- Some "files" in the /proc tree (such as ioports, interrupts, and tty/driver/serial).

Desired merged version, pages 1–2

Troubleshooting

My Modem is Physically There but Can't be Found

The error messages could be something like "No modem detected", "Modem not responding", or (strange) "You are already online" (from Minicom). If you have installed an internal modem (serial port is builtin) or are using an external one and don't know what serial port it's connected to then the problem is to find the serial port. See *My Serial Port is Physically There but Can't be Found*. This section is about finding out which serial port has the modem on it.

There's a program that looks for modems on commonly used serial ports called "wvdialconf". Just type "wvdialconf <a−new−file−name>". It will create the new file as a configuration file but you don't need this file unless you are going to use "wvdial" for dialing. See What is wvdialconf? Unfortunately, if your modem is in "online data" mode, wvdialconf will report "No modem detected" See No response to AT

Your problem could be due to a winmodem (or the like) which usually can't be used with Linux. The "setserial program may be used to detect serial ports but will not detect modems on them. Thus "wvdialconf" is best to try first.

Another way try to find out if there's a modem on a port is to start "minicom" on the port (after first setting up minicom for the correct serial port − you will need to save the setup and then exit minicom and start it again). Then type "AT" and you should see OK (or 0 if it's set for "digit result codes"). The results may be:

- No response. See *No response to AT*
- It takes many seconds to get an expected truncated response (including only the cursor moving down one line). See Extremely Slow: Text appears on the screen slowly after long delays
- Some strange characters appear but they are not in response to AT. This likely means that your modem is still connected to something at the other end of the phone line which is sending some cryptic packets or the like.

No response to AT

The modem should send you "OK" in response to your "AT" which you type to the modem (using minicom or the like). If you don't see "OK" (and in most cases don't even see the "AT" you typed either) then the modem is not responding (often because what you type doesn't even get to the modem).

A common cause is that there is no modem on the serial port you are typing to. For the case of an internal modem, that serial port likely doesn't exist either. That's because the PnP modem card (which has a built−in serial port) has either not been configured or has been configured incorrectly. See My Serial Port is Physically There but Can't be Found.

If what you type is really getting thru to a modem, then the lack of response could be due to the modem being in "online data" mode where it can't accept any AT commands. You may have been using the modem and then abruptly disconnected (such as killing the process with signal 9). In that case your modem did not get reset to "command mode" where it can interact to AT commands. Thus the message from minicom "You are already online. Hangup first." Well, you are sort of online but you are may not be connected to anything over the phone line. Wvdial will report "modem not responding" for the same situation.

To fix this as a last resort you could reboot the computer. Another way to try to fix this is to send +++ to the modem to tell it to escape back to "command mode" from "online data mode". On both sides of the +++ sequence there must be about 1 second of delay (nothing sent during "guard time"). This may not work if another process is using the modem since the +++ sequence could wind up with other characters inserted in between them or after the +++ (during the guard time). Ironically, even if the modem line is idle, typing an unexpected +++ is likely to set off an exchange of control packets (that you never see) that will violate the required guard time so that the +++ doesn't do what you wanted. +++ is usually in the string that is named "hangup string" so if you command minicom (or the like) to hangup it might work. Another way to do this is to just exit minicom and then run minicom again.

My Serial Port is Physically There but Can't be Found

If a physical device (such as a modem) doesn't work at all it may mean that the device is not at the I/O address that setserial thinks it's at. It could also mean (for a PnP card) that is doesn't yet have an address. Thus it can't be found.

Check the BIOS menus and BIOS messages. For the PCI bus use lspci or scanpci. If it's an ISA bus PnP serial port, try "pnpdump −−dumpregs" and/or see Plug−and−Play−HOWTO. Using "scanport" will scan all ISA bus ports and may discover an unknown port that could be a serial port (but it doesn't probe the port). It could hang your PC. You may try probing with setserial. See Probing. If nothing seems to get thru the port it may be accessible but have a bad interrupt. See Extremely Slow: Text appears on the screen slowly after long delays. Use setserial −g to see what the serial driver thinks and check for IRQ and I0 address conflicts. Even if you see no conflicts the driver may have incorrect information (view it by "setserial" and conflicts may still exist.

If two ports have the same IO address then probing it will erroneously indicate only one port. Plug−and−play detection will find both ports so this should only be a problem if at least one port is not plug−and−play. All sorts of errors may be reported/observed for devices illegally "sharing" a port but the fact that there are two devices on the same a port doesn't seem to get detected (except hopefully by you). In the above case, if the IRQs are different then probing for IRQs with setserial might "detect" this situation by failing to detect any IRQ. See Probing.

"Modem is busy"

What this means depends on what program sent it. The modem could actually be in use (busy). Another cause reported for the SuSE distribution is that there may be two serial drivers present instead of one. One driver was built into the kernel and the second was a module.

In kppp, this message is sent when an attempt to get/set the serial port parameters fails. These parameters are the one you see if you give the `stty -a` command. It's similar to the "Input/output error" one may get when trying to use `stty -F /dev/ttyS`*x*. Although the port is already opened when you see this message, I think it's possible to open a non−existent device (or one with the wrong IRQ or IO address). Now getting certain port parameters (such as the speed) means communicating with the serial port hardware. So this error message may mean that there is no serial port there (or that the modem's serial port has an incorrect (or no) IRQ or I0 address. It could be a wrong ttySx number. If /dev/modem is used it should be linked to the correct ttySx. In these cases the error message should have said: "Modem can't be found" which really means that it's serial port (often built into the modem) can't be found.

I can't get near 56k on my 56k modem

There must be very low noise on the line for it to work at even close to 56k. Some phone lines are so bad that the speeds obtainable are much slower than 56k (like 28.8k or even slower). Sometimes extension phones connected to the same line can cause problems. To test this you might connect your modem directly at the point where the telephone line enters the building with the feeds for everything else on that line disconnected (if others can tolerate such a test).

Uploading (downloading) files is broken/slow

Flow control (both at your PC and/or modem−to−modem) may not be enabled. For the uploading case: If you have set a high DTE speed (like 115.2k) then flow from your modem to your PC may work OK but uploading flow in the other direction will not all get thru due to the telephone line bottleneck. This will result in many errors and the resending of packets. It may thus take far too long to send a file. In some cases, files don't make it thru at all

Desired merged version, pages 3–4

For the downloading case: If you're downloading long uncompressed files or web pages (and your modem uses data compression) or if you've set a low DTE speed, then downloading may also be broken due to no flow control.

I Keep Getting: ''Id "S3" respawning too fast: disabled for 5 minutes''

Id "S3" is just an example. In this case look on the line which starts with "S3" in /etc/inittab and calls getty. This line causes the problem. Make sure the syntax for this line is correct and that the device (ttyS3) exists and can be found. If the modem has negated CD and getty opens the port, you'll get this error message since negated CD will kill getty. Then getty will respawn only to be killed again, etc. Thus it respawns over and over (too fast). It seems that if the cable to the modem is disconnected or you have the wrong serial port, it's just like CD is negated. All this can occur when your modem is chatting with getty. Make sure your modem is configured correctly. Look at AT commands E and Q.

If you use uugetty, verify that your /etc/gettydefs syntax is correct by doing the following:

```
linux# getty -c /etc/gettydefs
```

This can also happen when the uugetty initialization is failing. See section uugetty Still Doesn't Work.

My Modem is Hosed after Someone Hangs Up, or uugetty doesn't respawn

This can happen when your modem doesn't reset when DTR is dropped. Greg Hankins saw his RD and SD LEDs go crazy when this happened. You need to have your modem reset. Most Hayes compatible modems do this with &D3, but for USR Courier, SupraFax, and other modems, you must set &D2 (and S13=1 for USR Courier). Check your modem manual (if you have one).

uugetty Still Doesn't Work

There is a DEBUG option that comes with getty_ps. Edit your config file /etc/conf.{uu}getty.ttySN and add DEBUG=NNN. Where NNN is one of the following combination of numbers according to what you are trying to debug:

```
D_OPT   001          option settings
D_DEF   002          defaults file processing
D_UTMP  004          utmp/wtmp processing
D_INIT  010          line initialization (INIT)
D_GTAB  020          gettytab file processing
D_RUN   040          other runtime diagnostics
D_RB    100          ringback debugging
D_LOCK  200          uugetty lockfile processing
D_SCH   400          schedule processing
D_ALL   777          everything
```

Setting DEBUG=010 is a good place to start.

If you are running syslogd, debugging info will appear in your log files. If you aren't running syslogd info will appear in `/tmp/getty:ttySN` for debugging getty and `/tmp/uugetty:ttySN` for uugetty, and in `/var/adm/getty.log`. Look at the debugging info and see what is going on. Most likely, you will need to tune some of the parameters in your config file, and reconfigure your modem.

You could also try mgetty. Some people have better luck with it.

Extremely Slow: Text appears on the screen slowly after long delays

It's likely mis–set/conflicting interrupts. Here are some of the symptoms which will happen the first time you try to use a modem, terminal, or serial printer. In some cases you type something but nothing appears on the screen until many seconds later. Only the last character typed may show up. It may be just an invisible <return> character so all you notice is that the cursor jumps down one line. In other cases where a lot of data should appear on the screen, only a batch of about 16 characters appear. Then there is a long wait of many seconds for the next batch of characters. You might also get "input overrun" error messages (or find them in logs).

If it involves Plug–and–Play devices, see also Plug–and–Play–HOWTO.

As a quick check to see if it really is an interrupt problem, set the IRQ to 0 with "setserial". This will tell the driver to use polling instead of interrupts. If this seems to fix the "slow" problem then you had an interrupt problem. You should still try to solve the problem since polling uses excessive computer resources.

Checking to find the interrupt conflict may not be easy since Linux supposedly doesn't permit any interrupt conflicts and will send you a /dev/ttyS?: Device or resource busy error message if it thinks you are attempting to create a conflict. But a real conflict can be created if "setserial" has told the kernel incorrect info. The kernel has been lied to and thus doesn't think there is any conflict. Thus using "setserial" will not reveal the conflict (nor will looking at /proc/interrupts which bases

its info on "setserial"). You still need to know what "setserial" thinks so that you can pinpoint where it's wrong and change it when you determine what's really set in the hardware.

What you need to do is to check how the hardware is set by checking jumpers or using PnP software to check how the hardware is actually set. For PnP run either "pnpdump ––dumpregs" (if ISA bus) or run "lspci" (if PCI bus). Compare this to how Linux (e.g. "setserial") thinks the hardware is set.

Somewhat Slow: I expected it to be a few times faster

One reason may be that whatever is on the serial port (such as a modem, terminal, printer) doesn't work as fast as you thought it did. A 56k Modem seldom works at 56k and the Internet often has congestion and bottlenecks that slow things down. If the modem on the other end does not have a digital connection to the phone line (and uses a special "digital modem" not sold in most computer stores), then speeds above 33.6k are not possible.

Use "setserial –g /dev/ttyS*". If it shows anything less than a 16550A, this may be your problem. If you think that "setserial" has it wrong check it out. See What is Setserial for more info. If you really do have an obsolete serial port, lying about it to setserial will only make things worse.

The Startup Screen Show Wrong IRQs for the Serial Ports.

Linux does not do any IRQ detection on startup. When the serial module loads it only does serial device detection. Thus, disregard what it says about the IRQ, because it's just assuming the standard IRQs. This is done, because IRQ detection is unreliable, and can be fooled. But if and when setserial runs from a start–up script, it changes the IRQ's and displays the new (and hopefully correct) state on on the startup screen. If the wrong IRQ is not corrected by a later display on the screen, then you've got a problem.

So, even though I have my ttyS2 set at IRQ 5, I still see

ttyS02 at 0x03e8 (irq = 4) is a 16550A

at first when Linux boots. (Older kernels may show "ttyS02" as "tty02" which is the same as ttyS2). You may need to use setserial to tell Linux the IRQ you are using.

"Cannot create lockfile. Sorry"

When a port is "opened" by a program a lockfile is created in /var/lock/. Wrong permissions for the lock directory will not allow a lockfile to be created there. Use "ls –ld /var/lock" to see if the permissions are OK: usually rwx for everyone (repeated 3 times). If it's wrong, use "chmod" to fix it. Of course, if there is no "lock" directory no lockfile can be created there. For more info on lockfiles see the Serial–HOWTO subsection:"What Are Lock Files".

"Cannot open /dev/ttyS?: Permission denied"

Check the file permissions on this port with "ls –l /dev/ttyS?"_ If you own the ttyS? then you need read and write permissions: crw with the c (Character device) in col. 1. It you don't own it then it should show rw– in cols. 8 & 9 which means that everyone has read and write permission on it. Use "chmod" to change

Desired merged version, pages 5–6

permissions. There are more complicated ways to get access like belonging to a "group" that has group permission.

"Operation not supported by device" for ttyS?

This means that an operation requested by setserial, stty, etc. couldn't be done because the kernel doesn't support doing it. Formerly this was often due to the "serial" module not being loaded. But with the advent of PnP, it may likely mean that there is no modem (or other serial device) at the address where the driver (and setserial) thinks it is. If there is no modem there, commands (for operations) sent to that address obviously don't get done. See What is set in my serial port hardware?

If the "serial" module wasn't loaded but "lsmod" shows you it's now loaded it might be the case that it's loaded now but wasn't loaded when you got the error message. In many cases the module will automatically loaded when needed (if it can be found). To force loading of the "serial" module it may be listed in the file: /etc/modules.conf or /etc/modules. The actual module should reside in: /lib/modules/.../misc/serial.o.

"Device /dev/ttyS? is locked."

This means that someone else (or some other process) is supposedly using the serial port. There are various ways to try to find out what process is "using" it. One way is to look at the contents of the lockfile (/var/lock/LCK...). It should be the process id. If the process id is say 100 type "ps 100" to find out what it is. Then if the process is no longer needed, it may be gracefully killed by "kill 100". If it refuses to be killed use "kill −9 100" to force it to be killed, but then the lockfile will not be removed and you'll need to delete it manually. Of course if there is no such process as 100 then you may just remove the lockfile but in most cases the lockfile should have been automatically removed if it contained a stale process id (such as 100).

"/dev/tty? Device or resource busy"

This means that the device you are trying to access (or use) is supposedly busy (in use) or that a resource it needs (such as an IRQ) is supposedly being used by another device (the resource is "busy"). This message is easy to understand if it only means that the device is busy (in use). But it often means that a resource is in use. What makes it even more confusing is that in some cases neither the device not the resources that it needs are actually "busy".

The ‘‘resource busy’’ part often means (example for ttyS2) ‘‘You can't use ttyS2 since another device is using ttyS2's interrupt.’’ The potential interrupt conflict is inferred from what "setserial" thinks. A more accurate error message would be ‘‘Can't use ttyS2 since the setserial data (and kernel data) indicates that another device is using ttyS2's interrupt’’. If two devices use the same IRQ and you start up only one of the devices, everything is OK because there is no conflict yet. But when you next try to start the second device (without quitting the first device) you get a "... busy" error message. This is because the kernel only keeps track of what RQs are actually in use and actual conflicts don't happen unless the devices are in use (open). The situation for I/O address (such as 0x3f8) conflict is similar.

This error is sometimes due to having two serial drivers: one a module and the other compiled into the kernel. Both drivers try to grab the same resources and one driver finds them "busy".

There are two possible cases when you see this message:

1. There may be a real resource conflict that is being avoided.
2. Setserial has it wrong and the only reason ttyS2 can't be used is that

   setserial erroneously predicts a conflict.

What you need to do is to find the interrupt setserial thinks ttyS2 is using. Look at /proc/tty/driver/serial (if you have it). You should also be able to find it with the "setserial" command for ttyS2. But due to a bug (reported by me in Nov. 2000) you get the same "... busy" error message when you try this with "setserial".

To try to resolve this problem reboot or: exit or gracefully kill all likely conflicting processes. If you reboot: 1. Watch the boot−time messages for the serial ports. 2. Hope that the file that runs "setserial" at boot−time doesn't (by itself) create the same conflict again.

If you think you know what IRQ say ttyS2 is using then you may look at /proc/interrupts to find what else (besides another serial port) is currently using this IRQ. You might also want to double check that any suspicious IRQs shown here (and by "setserial") are correct (the same as set in the hardware). A way to test whether or not it's a potential interrupt conflict is to set the IRQ to 0 (polling) using "setserial". Then if the busy message goes away, it was likely a potential interrupt conflict. It's not a good idea to leave it permanently set at 0 since it will make the CPU work too hard.

"Input/output error" from setserial or stty

You may have typed "ttys" instead of "ttyS". You will see this error message if you try to use the setserial command for any device that is not a serial port. It also may mean that the serial port is in use (busy or opened) and thus the attempt to get/set parameters by setserial or stty failed. It could also mean that there isn't any serial port at the IO address that setserial thinks your port is at.

Overrun errors on serial port

This is an overrun of the hardware FIFO buffer and you can't increase its size. See "Higher Serial Thruput" in the Serial−HOWTO.

Modem doesn't pick up incoming calls

This paragraph is for the case where a modem is used for both dial−in and dial−out. If the modem generates a DCD (=CD) signal, some programs (but not mgetty) will think that the modem is busy. This will cause a problem when you are trying to dial out with a modem and the modem's DCD or DTR are not

implemented correctly. The modem should assert DCD only hen there is an actual connection (ie someone has dialed in), not when getty is watching the

port. Check to make sure that your modem is configured to only assert DCD when there is a connection (&C1). DTR should be on (asserted) by the

communications program whenever something is using, or watching the line, like getty, kermit, or some other comm program.

Troubleshooting Tools

These are some of the programs you might want to use in troubleshooting:

- "lsof /dev/ttyS*" will list serial ports which are open.
- "setserial" shows and sets the low−level hardware configuration of a port (what the driver thinks it is). See What is Setserial
- "stty" shows and sets the configuration of a port (except for that handled by "setserial"). See the Serial−HOWTO section: "Stty".
- "modemstat" or "statserial" will show the current state of various modem signal lines (such as DTR, CTS, etc.)
- "irqtune" will give serial port interrupts higher priority to improve performance.
- "hdparm" for hard−disk tuning may help some more.
- "lspci" shows the actual IRQs, etc. of hardware on the PCI bus.
- "pnpdump −−dumpregs" shows the actual IRQs, etc. of hardware for PnP devices on the ISA bus.
- Some "files" in the /proc tree (such as ioports, interrupts, and tty/driver/serial).

Appendix F

Use Case 5: Maintaining Several Versions of Web Pages

Contents

XHTML source for base complex version, rendering in section 2.5 157
XHTML source for simple base version 159
Rendering of updated simple version 160
Rendering of manually merged complex version 160
XHTML source for updated simple version 161
XHTML source for merged complex version 161

The source listings (SVG and XHTML) have been shortened to occupy less space. The ellipsis indicate omissions in the form of shortened paragraphs.

ccs-complex.html **Page 1 of 4**

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta name="generator" content="HTML Tidy, see www.w3.org" />
  <title>The Chaotic Chess Society</title>
  <meta name="GENERATOR" content="Microsoft FrontPage 3.0" />
 </head>
 <body>
  <div align="center">
   <center>
    <table border="0" cellpadding="0" cellspacing="0" width="98%">
     <tr>
      <td align="right" valign="top" width="20%" bgcolor="#C0C0C0">
       </td>
      <td width="15" bgcolor="#808000"> </td>
      <td valign="bottom" width="80%" bgcolor="#C0C0C0">
       <big>
        <big>
         <big>
          <font face="Arial">The Chaotic Chess Society</font>
         </big>
        </big>
       </big>
      </td>
     </tr>
     <tr>
      <td valign="top" width="20%" bgcolor="#808000">
       <small>
        <small>
         <font face="Arial">
          <strong>Chess Links (from the FAQ)</strong>
         </font>
        </small>
       </small>
       <ul align="left">
        <li>
         <small>
          <a href="http://www.clark.net/pub/pribut/chess.html">Steve Pribut's
          Chess Page</a>
         </small>
        </li>
        <li>
         <small>
          <a href="http://www.chesscenter.com/twic/twic.html">The Week In
          Chess</a>
         </small>
        </li>
        <li>
         <small>
         <a href="http://home.earthlink.net/~kaechster/play.html">Play Chess
         Online</a>
         Listing of places to play chess online</small>
        </li>
        <li>
         <small>
          <a href="http://www.chesscenter.com/coaching.html">Online Coachin
g
          Sites</a>
         </small>
        </li>
        <li>
         <small>
          <a href="http://www.traveller.com/chess/">WWW Chess Archives</a>
         </small>
        </li>
        <li>
         <small>
          <a href="http://insidechess.com/">Inside Chess Magazine</a>
         </small>
        </li>
        <li>
         <small>
          <a href="http://www.xs4all.nl/~timkr/chess/chess.html">Tim
```

```
   Krabb&#233; Chess Curiosities</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.kasparovchess.com/">Kasparov Chess</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.cais.com/sunburst/chess/">Jerry Lawson's Chess
   Pages(USA, DC, World, US Chess Center Info)</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://caissa.onenet.net/chess/">Internet Chess
   Library</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://caissa.onenet.net/chess/HTML/offsite.html">Chess
   sites</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.chemeng.ed.ac.uk/people/steve/">British Chess
   Links Page</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.ub.uit.no/chess/">CHESS: Rudof Steinkellner,
   Jr.</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.chesscanada.org/">Chess Federation of
   Canada</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.chessclub.com/">Internet Chess Club</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.chess.net/">- Chess.Net Live Chess</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.eics.com/iecg/index.html">IECG Website</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.rebel.nl/">Schroder BV Software - Home of
   Rebel</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.princeton.edu/~jedwards/cif/intro.html">Chess
   Primer-Jon Edwards</a>
  </small>
 </li>
 <li>
  <small>
   <a href="http://www.chesscafe.com/">Chess Cafe - Russell Hanon</a>
  </small>
```



```
        </li>
       </ul>
       <p>
        <font size="3" face="Arial">
         <br />
        </font>
       </p>
      </td>
      <td width="15" bgcolor="#C0C0C0">    &nbs
p;</td>
      <td valign="top" width="80%">
       <table border="0" bgcolor="#C0C0C0" width="100%" cellspacing="0"
       cellpadding="0">
        <tr>
         <td bgcolor="#808000"> </td>
         <td> </td>
         <td bgcolor="#808000"> </td>
         <td> </td>
         <td bgcolor="#808000"> </td>
         <td> </td>
         <td bgcolor="#808000"> </td>
         <td> </td>
         <td bgcolor="#808000"> </td>
        </tr>
        <tr>
         <td> </td>
         <td bgcolor="#808000"> </td>
         <td> </td>
         <td bgcolor="#808000"> </td>
         <td> </td>
         <td bgcolor="#808000"> </td>
         <td> </td>
         <td bgcolor="#808000"> </td>
         <td> </td>
        </tr>
       </table>
       <p align="center">
        <font size="3">
         <img src="chess1.gif" width="160" height="120"
         alt="chess1.gif (16254 bytes)" />
         <br />
        </font>
       </p>
       <p align="center">
        <big>The
        <big>
         <big>
          <em>CCS</em>
         </big>
        </big>
        is a small chess club in southern rural Finland.</big>
       </p>
       <table border="0" width="698" cellspacing="0" cellpadding="0">
        <tr>
         <td width="344" valign="top" bgcolor="#808000">
          <h3>Our members:</h3>
          <ul>
           <li>
            <font face="Century Gothic">Jussi Virtanen (chairman)</font>
           </li>
           <li>
            <font face="Century Gothic">Anna Virtanen</font>
           </li>
           <li>
            <font face="Century Gothic">Petteri Ilmala</font>
           </li>
           <li>
            <font face="Century Gothic">Anna-Kaisa Pokkanaama</font>
           </li>
          </ul>
         </td>
         <td width="342" bgcolor="#C0C0C0">
          <h3>Contact Information</h3>
          <p>Please Write to</p>
          <p>
```



```
           <tt>Jussi Virtanen
           <br />
           Katukatu 12 A 4
           <br />
           02311 Yl&#228;sein&#228;joki
           <br />
           Finland</tt>
          </p>
          <p>You can also contatct us a
          <a href="mailto:jussi.virtanen@ccs.kone.domaini.fi">
           <tt>jussi.virtanen@ccs.kone.domaini.fi</tt>
          </a>
          </p>
         </td>
        </tr>
       </table>
       <h2>
        <font face="Arial">About Chess</font>
       </h2>
       <p>Here is some information about chess in general (from the Chess FAQ
       by Steve Pribut).</p>
       <h3 align="left">
        <small>I'm a Novice (or Intermediate). How Do I Improve?</small>
       </h3>
       <p align="left">
        <small>There are lots of variations ... them.</small>
       </p>
       <p align="left">
        <small>Some books are listed below to ... endgame book.</small>
       </p>
       <p align="left">
        <small>You should also consider ... games.</small>
       </p>
       <p align="left">
        <small>If you are web oriented check ... Edwards</small>
       </p>
       <h3 align="left">
        <small>Using Graphic Chess Symbols in Printed Text</small>
       </h3>
       <p>
        <small>There are a few ways of composing chess ... including diagrams
        in printed text):</small>
       </p>
       <ol>
        <li>
         <small>Use a word processor or page-layout ... ).</small>
        </li>
        <li>
         <small>Use a chess-specific writing ... proprietary.</small>
        </li>
        <li>
         <small>CC-Publisher (MS Windows) is another ...
         mginsbur@rnd.stern.nyu.edu</small>
        </li>
        <li>
         <small>Use the LaTeX chess macros and fonts ... effort.</small>
        </li>
       </ol>
       <p align="center">
        <font size="3">.</font>
       </p>
      </td>
     </tr>
    </table>
   </center>
  </div>
  <hr />
  <p>
   <em>This page was last updated $Id: ccs-complex.html,v 1.3 2001/09/08
   15:55:18 ctl Exp $</em>
  </p>
 </body>
</html>
```

ccs-simple.html Page 1 of 2

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta name="generator" content="HTML Tidy, see www.w3.org" />
  <title>The Chaotic Chess Society</title>
  <meta name="GENERATOR" content="Microsoft FrontPage 3.0" />
 </head>
 <body>
  <h1>The Chaotic Chess Society</h1>
  <p>The CCS is a small chess club in southern rural Finland. Our members
  are</p>
  <ul>
   <li>Jussi Virtanen (chairman)</li>
   <li>Anna Virtanen</li>
   <li>Petteri Ilmala</li>
   <li>Anna-Kaisa Pokkanaama</li>
  </ul>
  <h3>Contact Information</h3>
  <p>Please Write to</p>
  <p>
   <tt>Jussi Virtanen
   <br />
   Katukatu 12 A 4
   <br />
   02311 Yl&#228;sein&#228;joki
   <br />
   Finland</tt>
  </p>
  <p>You can also contatct us a
  <a href="mailto:jussi.virtanen@ccs.kone.domaini.fi">
   <tt>jussi.virtanen@ccs.kone.domaini.fi</tt>
  </a>
  </p>
  <h2>About Chess</h2>
  <p>Here is some information about chess in general (from the Chess FAQ by
  Steve Pribut).</p>
  <h3 align="left">I'm a Novice (or Intermediate). How Do I Improve?</h3>
  <p align="left">There are lots of variations to the methods, ... games
  against stronger players, and learn from them.</p>
  <p align="left">Some books are listed below to help in the ... book or two,
  and an endgame book.</p>
  <p align="left">You should also consider reviewing classical ... well
  annotated games.</p>
  <p align="left">If you are web oriented check the site of Jon ...
  Edwards</p>
  <h3 align="left">Using Graphic Chess Symbols in Printed Text</h3>
  <p>There are a few ways of composing chess texts in ... printed text):</p>
  <ol>
   <li>Use a word processor or page-layout program and a chess ...
   ("mailto:anw@maths.nott.ac.uk"anw@maths.nott.ac.uk ).</li>
   <li>Use a chess-specific writing application. ChessWriter ... ChessWriter
   is proprietary.</li>
   <li>CC-Publisher (MS Windows) is another commercial ...
   mginsbur@rnd.stern.nyu.edu</li>
   <li>Use the LaTeX chess macros and fonts package by Piet ... but the
   results are worth the effort.</li>
  </ol>
  <h2>Chess Links (from the FAQ)</h2>
  <ul align="left">
   <li>
    <a href="http://www.clark.net/pub/pribut/chess.html">Steve Pribut's Ches
s
    Page</a>
   </li>
   <li>
    <a href="http://www.chesscenter.com/twic/twic.html">The Week In Chess
</a>
   </li>
   <li>
   <a href="http://home.earthlink.net/˜kaechster/play.html">Play Chess
   Online</a>
   Listing of places to play chess online</li>
   <li>
```

ccs-simple.html Page 2 of 2

```
    <a href="http://www.chesscenter.com/coaching.html">Online Coaching
    Sites</a>
   </li>
   <li>
    <a href="http://www.traveller.com/chess/">WWW Chess Archives</a>
   </li>
   <li>
    <a href="http://insidechess.com/">Inside Chess Magazine</a>
   </li>
   <li>
    <a href="http://www.xs4all.nl/˜timkr/chess/chess.html">Tim Krabb&#233;
    Chess Curiosities</a>
   </li>
   <li>
    <a href="http://www.kasparovchess.com/">Kasparov Chess</a>
   </li>
   <li>
    <a href="http://www.cais.com/sunburst/chess/">Jerry Lawson's Chess
    Pages(USA, DC, World, US Chess Center Info)</a>
   </li>
   <li>
    <a href="http://caissa.onenet.net/chess/">Internet Chess Library</a>
   </li>
   <li>
    <a href="http://caissa.onenet.net/chess/HTML/offsite.html">Chess
    sites</a>
   </li>
   <li>
    <a href="http://www.chemeng.ed.ac.uk/people/steve/">British Chess Links
    Page</a>
   </li>
   <li>
    <a href="http://www.ub.uit.no/chess/">CHESS: Rudof Steinkellner, Jr.</a>
   </li>
   <li>
    <a href="http://www.chesscanada.org/">Chess Federation of Canada</a>
   </li>
   <li>
    <a href="http://www.chessclub.com/">Internet Chess Club</a>
   </li>
   <li>
    <a href="http://www.chess.net/">- Chess.Net Live Chess</a>
   </li>
   <li>
    <a href="http://www.eics.com/iecg/index.html">IECG Website</a>
   </li>
   <li>
    <a href="http://www.rebel.nl/">Schroder BV Software - Home of Rebel</a>
   </li>
   <li>
    <a href="http://www.princeton.edu/˜jedwards/cif/intro.html">Chess
    Primer-Jon Edwards</a>
   </li>
   <li>
    <a href="http://www.chesscafe.com/">Chess Cafe - Russell Hanon</a>
   </li>
  </ul>
  <hr />
  <p>
   <em>This page was last updated $Id: ccs-simple.html,v 1.2 2001/09/08
   15:55:19 ctl Exp $</em>
  </p>
 </body>
</html>
```

The Chaotic Chess Society

The CCS is a small chess club in southern rural Finland. Our members are

- Jussi Virtanen *(chairman)*
- Anna Virtanen
- Petteri Ilmala
- Anna-Kaisa Pokkanaama *(secretary)*

Contact Information

Please Write to

```
Jussi Virtanen
Katukatu 12 A 4
02311 Yläseinäjoki
Finland
```

You can also contatct us a jussi.virtanen@ccs.kone.domaini.fi

About Chess

Here is some information about chess in general (from the Chess FAQ by Steve Pribut). The FAQ can be found here

I'm a Novice (or Intermediate). How Do I Improve?

There are lots of variations to the methods, but the things most good teachers agree on is to emphasize (1) tactics, (2) endings, and (3) playing with a plan. Most people spend too much time studying openings. Just learn enough about openings to get to a playable middlegame. The books listed below should give you a great start on (1), (2), and (3). Of course, playing experience is important. Review your games (with a much stronger player if possible) or your chess computer to find out what you did right and wrong. Seek out games against stronger players, and learn from them.

Some books are listed below to help in the quest to improve. You don't need to buy all these--pick and choose as you please. Buy one or two general works, a tactics book or two, and an endgame book.

You should also consider reviewing classical games by the masters: Capablanca, Tal, and others. Read over well annotated games.

If you are web oriented check the site of Jon Edwards, U.S. Correspondance Chess Champion's Home Page. This has a lot of introductory material on learning to play chess, some tactics and openings. Chess Primer & Intro To ChessJon Edwards

Using Graphic Chess Symbols in Printed Text

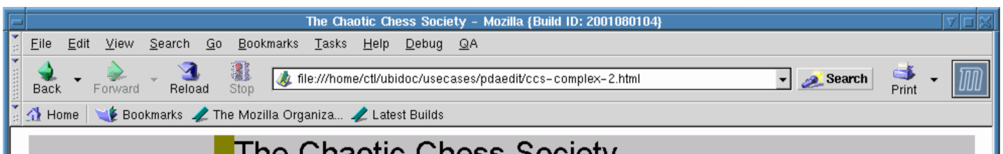
There are a few ways of composing chess texts in international figurine notation (or including diagrams in printed text):

1. Use a word processor or page-layout program and a chess font. For instance, for the Apple Macintosh there are at least 3 different sets of fonts usable with standard word processors like Microsoft Word, MacWrite, Nisus or WriteNow; or with page-layout programs like Illustrator or PageMaker. Most of these fonts are proprietary (you must purchase them). The fonts usually can be used for both the figurines and the diagrams. A freely available/usable PostScript font, including a variety of figurines, diagrams and _Informant_ symbols, has been posted to "news:comp.fonts"comp.fonts and "news:rec.games.chess"rec.games.chess by Andy Walker ("mailto:anw@maths.nott.ac.uk"anw@maths.nott.ac.uk).
2. CC-Publisher (MS Windows) is another commercial chess-specific writing application. You must have MS Windows, a word processing package (Word, WordPerfect, AmiPro), and a chess database system (for generating diagrams--although this could be done by hand--like ChessBase or Zarkov). It comes in two versions. The basic version supports HP LJ-compatible laserjet printers ($49.95). The deluxe version supports any PostScript printer, and comes with PostScript Type I or TrueType fonts ($139.95). You get integrated utilities to move you from game-entry or diagram-creation to conversion and import into your word processor, with special Tips and Tricks for MS Word, Lotus AmiPro, and WordPerfect users. Extremely easy installation, and your fonts become available to all Windows applications. There's a comprehensive user manual on the installation disk, and you get free technical support! Chess Chow Publications, P.O. Box 3348, Church St. Station, New York, NY 10008. 212-432-6546. e-mail mginsbur@rnd.stern.nyu.edu
3. Use the LaTeX chess macros and fonts package by Piet Tutelaers (see [18]). TeX is an advanced public-domain system for text formatting available on mainframes, workstations and personal computers. LaTeX is a set of text-formatting macros for TeX. METAFONT is a font generator program for TeX. For general information on all of these, and pointers to reference manuals, see the FAQ list posting in comp.text.tex.) Once you have the chess package, you'll need to 3a) be able to use METAFONT to generate chess fonts starting from the programs contained in the package; 3b) be able to install the LaTeX macros in your TeX system; and 3c) learn the macro language to format chess texts. Activity 3a can become tiresome if you do not have any help from a TeX wizard. Using LaTeX to write chess text is not very simple, but the results are worth the effort.

Chess Links (from the FAQ)

- Steve Pribut's Chess Page
- The Week In Chess
- Play Chess Online Listing of places to play chess online
- Online Coaching Sites
- WWW Chess Archives
- Inside Chess Magazine
- Tim Krabbé Chess Curiosities
- Kasparov Chess
- Jerry Lawson's Chess Pages(USA, DC, World, US Chess Center Info)
- Internet Chess Library
- Chess sites
- British Chess Links Page
- CHESS: Rudof Steinkellner, Jr.
- Chess Federation of Canada
- Chess.Net Live Chess
- Schroder BV Software - Home of Rebel
- Chess Primer-Jon Edwards
- Chess Cafe - Russell Hanon

This page was last updated $Id: ccs-simple-2.html,v 1.2 2001/09/08 15:55:19 ctl Exp $



ccs-simple-2.html Page 1 of 2

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta name="generator" content="HTML Tidy, see www.w3.org" />
  <title>The Chaotic Chess Society</title>
  <meta name="GENERATOR" content="Microsoft FrontPage 3.0" />
 </head>
 <body>
  <h1>The Chaotic Chess Society</h1>
  <p>The CCS is a small chess club in southern rural Finland. Our members
  are</p>
  <ul>
   <li>Jussi Virtanen
   <em>(chairman)</em>
   </li>
   <li>Anna Virtanen</li>
   <li>Petteri Ilmala</li>
   <li>Anna-Kaisa Pokkanaama
   <em>(secretary)</em>
   </li>
  </ul>
  <h3>Contact Information</h3>
  <p>Please Write to</p>
  <p>
   <tt>Jussi Virtanen
   <br />
   Katukatu 12 A 4
   <br />
   02311 Yl&#228;sein&#228;joki
   <br />
   Finland</tt>
  </p>
  <p>You can also contatct us a
  <a href="mailto:jussi.virtanen@ccs.kone.domaini.fi">
   <tt>jussi.virtanen@ccs.kone.domaini.fi</tt>
  </a>
  </p>
  <h2>About Chess</h2>
  <p>Here is some information about chess in general (from the Chess FAQ by
  Steve Pribut). The FAQ can be found
  <a href="http://www.clark.net/pub/pribut/chessfaq.html">here</a>
  </p>
  <h3 align="left">I'm a Novice (or Intermediate). How Do I Improve?</h3>
  <p align="left">There are lots of variations to the methods, ... games
  against stronger players, and learn from them.</p>
  <p align="left">Some books are listed below to help in the ... book or two,
  and an endgame book.</p>
  <p align="left">You should also consider reviewing classical ... well
  annotated games.</p>
  <p align="left">If you are web oriented check the site of Jon ...
  Edwards</p>
  <h3 align="left">Using Graphic Chess Symbols in Printed Text</h3>
  <p>There are a few ways of composing chess texts in ... printed text):</p>
  <ol>
   <li>Use a word processor or page-layout program and a chess ...
   ("mailto:anw@maths.nott.ac.uk"anw@maths.nott.ac.uk ).</li>
   <li>CC-Publisher (MS Windows) is another commercial ...
   mginsbur@rnd.stern.nyu.edu</li>
   <li>Use the LaTeX chess macros and fonts package by Piet ... but the
   results are worth the effort.</li>
  </ol>
  <h2>Chess Links (from the FAQ)</h2>
  <ul align="left">
   <li>
    <a href="http://www.clark.net/pub/pribut/chess.html">Steve Pribut's Ches
s
    Page</a>
   </li>
   <li>
    <a href="http://www.chesscenter.com/twic/twic.html">The Week In Chess
</a>
   </li>
   <li>
```

ccs-simple-2.html Page 2 of 2

```
    <a href="http://home.earthlink.net/~kaechster/play.html">Play Chess
    Online</a>
    Listing of places to play chess online</li>
   <li>
    <a href="http://www.chesscenter.com/coaching.html">Online Coaching
    Sites</a>
   </li>
   <li>
    <a href="http://www.traveller.com/chess/">WWW Chess Archives</a>
   </li>
   <li>
    <a href="http://insidechess.com/">Inside Chess Magazine</a>
   </li>
   <li>
    <a href="http://www.xs4all.nl/~timkr/chess/chess.html">Tim Krabb&#233;
    Chess Curiosities</a>
   </li>
   <li>
    <a href="http://www.kasparovchess.com/">Kasparov Chess</a>
   </li>
   <li>
    <a href="http://www.cais.com/sunburst/chess/">Jerry Lawson's Chess
    Pages(USA, DC, World, US Chess Center Info)</a>
   </li>
   <li>
    <a href="http://caissa.onenet.net/chess/">Internet Chess Library</a>
   </li>
   <li>
    <a href="http://caissa.onenet.net/chess/HTML/offsite.html">Chess
    sites</a>
   </li>
   <li>
    <a href="http://www.chemeng.ed.ac.uk/people/steve/">British Chess Links
    Page</a>
   </li>
   <li>
    <a href="http://www.ub.uit.no/chess/">CHESS: Rudof Steinkellner, Jr.</a>
   </li>
   <li>
    <a href="http://www.chesscanada.org/">Chess Federation of Canada</a>
   </li>
   <li>
    <a href="http://www.chessclub.com/">Internet Chess Club</a>
   </li>
   <li>
    <a href="http://www.chess.net/">Chess.Net Live Chess</a>
   </li>
   <li>
    <a href="http://www.eics.com/iecg/index.html">IECG Website</a>
   </li>
   <li>
    <a href="http://www.rebel.nl/">Schroder BV Software - Home of Rebel</a>
   </li>
   <li>
    <a href="http://www.princeton.edu/~jedwards/cif/intro.html">Chess
    Primer-Jon Edwards</a>
   </li>
   <li>
    <a href="http://www.chesscafe.com/">Chess Cafe - Russell Hanon</a>
   </li>
  </ul>
  <hr />
  <p>
   <em>This page was last updated $Id: ccs-simple-2.html,v 1.2 2001/09/08
   15:55:19 ctl Exp $</em>
  </p>
 </body>
</html>
```

ccs-complex-2.html Page 1 of 4

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta name="generator" content="HTML Tidy, see www.w3.org" />
  <title>The Chaotic Chess Society</title>
  <meta name="GENERATOR" content="Microsoft FrontPage 3.0" />
 </head>
 <body>
  <div align="center">
   <center>
    <table border="0" cellpadding="0" cellspacing="0" width="98%">
     <tr>
      <td align="right" valign="top" width="20%" bgcolor="#C0C0C0">
       </td>
      <td width="15" bgcolor="#808000"> </td>
      <td valign="bottom" width="80%" bgcolor="#C0C0C0">
       <big>
        <big>
         <big>
          <font face="Arial">The Chaotic Chess Society</font>
         </big>
        </big>
       </big>
      </td>
     </tr>
     <tr>
      <td valign="top" width="20%" bgcolor="#808000">
       <small>
        <small>
         <font face="Arial">
          <strong>Chess Links (from the FAQ)</strong>
         </font>
        </small>
       </small>
       <ul align="left">
        <li>
         <small>
          <a href="http://www.clark.net/pub/pribut/chess.html">Steve Pribut's
          Chess Page</a>
         </small>
        </li>
        <li>
         <small>
          <a href="http://www.chesscenter.com/twic/twic.html">The Week In
          Chess</a>
         </small>
        </li>
        <li>
         <small>
         <a href="http://home.earthlink.net/~kaechster/play.html">Play Chess
         Online</a>
         Listing of places to play chess online</small>
        </li>
        <li>
         <small>
          <a href="http://www.chesscenter.com/coaching.html">Online Coachin
g
          Sites</a>
         </small>
        </li>
        <li>
         <small>
          <a href="http://www.traveller.com/chess/">WWW Chess Archives</a>
         </small>
        </li>
        <li>
         <small>
          <a href="http://insidechess.com/">Inside Chess Magazine</a>
         </small>
        </li>
        <li>
         <small>
          <a href="http://www.xs4all.nl/~timkr/chess/chess.html">Tim
```

ccs-complex-2.html **Page 2 of 4**

```
    Krabb&#233; Chess Curiosities</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.kasparovchess.com/">Kasparov Chess</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.cais.com/sunburst/chess/">Jerry Lawson's Chess
    Pages(USA, DC, World, US Chess Center Info)</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://caissa.onenet.net/chess/">Internet Chess
    Library</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://caissa.onenet.net/chess/HTML/offsite.html">Chess
    sites</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.chemeng.ed.ac.uk/people/steve/">British Chess
    Links Page</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.ub.uit.no/chess/">CHESS: Rudof Steinkellner,
    Jr.</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.chesscanada.org/">Chess Federation of
    Canada</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.chessclub.com/">Internet Chess Club</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.chess.net/">Chess.Net Live Chess</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.eics.com/iecg/index.html">IECG Website</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.rebel.nl/">Schroder BV Software - Home of
    Rebel</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.princeton.edu/˜jedwards/cif/intro.html">Chess
    Primer-Jon Edwards</a>
   </small>
  </li>
  <li>
   <small>
    <a href="http://www.chesscafe.com/">Chess Cafe - Russell Hanon</a>
   </small>
```

ccs-complex-2.html **Page 3 of 4**

```
      </li>
     </ul>
     <p>
      <font size="3" face="Arial">
       <br />
      </font>
     </p>
    </td>
    <td width="15" bgcolor="#C0C0C0">    &nbs
p;</td>
    <td valign="top" width="80%">
     <table border="0" bgcolor="#C0C0C0" width="100%" cellspacing="0"
     cellpadding="0">
      <tr>
       <td bgcolor="#808000"> </td>
       <td> </td>
       <td bgcolor="#808000"> </td>
       <td> </td>
       <td bgcolor="#808000"> </td>
       <td> </td>
       <td bgcolor="#808000"> </td>
       <td> </td>
       <td bgcolor="#808000"> </td>
      </tr>
      <tr>
       <td> </td>
       <td bgcolor="#808000"> </td>
       <td> </td>
       <td bgcolor="#808000"> </td>
       <td> </td>
       <td bgcolor="#808000"> </td>
       <td> </td>
       <td bgcolor="#808000"> </td>
       <td> </td>
      </tr>
     </table>
     <p align="center">
      <font size="3">
       <img src="chess1.gif" width="160" height="120"
       alt="chess1.gif (16254 bytes)" />
       <br />
      </font>
     </p>
     <p align="center">
      <big>The
      <big>
       <big>
        <em>CCS</em>
       </big>
      </big>
      is a small chess club in southern rural Finland.</big>
     </p>
     <table border="0" width="698" cellspacing="0" cellpadding="0">
      <tr>
       <td width="344" valign="top" bgcolor="#808000">
        <h3>Our members:</h3>
        <ul>
         <li>
          <font face="Century Gothic">Jussi Virtanen
          <em>(chairman)</em>
          </font>
         </li>
         <li>
          <font face="Century Gothic">Anna Virtanen</font>
         </li>
         <li>
          <font face="Century Gothic">Petteri Ilmala</font>
         </li>
         <li>
          <font face="Century Gothic">Anna-Kaisa Pokkanaama
          <em>(secretary)</em>
          </font>
         </li>
        </ul>
       </td>
```

ccs-complex-2.html **Page 4 of 4**

```
       <td width="342" bgcolor="#C0C0C0">
        <h3>Contact Information</h3>
        <p>Please Write to</p>
        <p>
         <tt>Jussi Virtanen
         <br />
         Katukatu 12 A 4
         <br />
         02311 Yl&#228;sein&#228;joki
         <br />
         Finland</tt>
        </p>
        <p>You can also contatct us a
        <a href="mailto:jussi.virtanen@ccs.kone.domaini.fi">
         <tt>jussi.virtanen@ccs.kone.domaini.fi</tt>
        </a>
        </p>
       </td>
      </tr>
     </table>
     <h2>
      <font face="Arial">About Chess</font>
     </h2>
     <p>Here is some information about chess in general (from the Chess FAQ
     by Steve Pribut). The FAQ can be found
     <a href="http://www.clark.net/pub/pribut/chessfaq.html">here</a>
     </p>
     <h3 align="left">
      <small>I'm a Novice (or Intermediate). How Do I Improve?</small>
     </h3>
     <p align="left">
      <small>There are lots of variations ... them.</small>
     </p>
     <p align="left">
      <small>Some books are listed below to ... endgame book.</small>
     </p>
     <p align="left">
      <small>You should also consider ... games.</small>
     </p>
     <p align="left">
      <small>If you are web oriented check ... Edwards</small>
     </p>
     <h3 align="left">
      <small>Using Graphic Chess Symbols in Printed Text</small>
     </h3>
     <p>
      <small>There are a few ways of composing chess ... including diagrams
      in printed text):</small>
     </p>
     <ol>
      <li>
       <small>Use a word processor or page-layout ... ).</small>
      </li>
      <li>
       <small>CC-Publisher (MS Windows) is another ...
       mginsbur@rnd.stern.nyu.edu</small>
      </li>
      <li>
       <small>Use the LaTeX chess macros and fonts ... effort.</small>
      </li>
     </ol>
     <p align="center">
      <font size="3">.</font>
     </p>
    </td>
   </tr>
  </table>
 </center>
 </div>
 <hr />
 <p>
  <em>This page was ...</em>
 </p>
</body>
</html>
```

Appendix G

Merge Cases

The cases are identified by a letter and a number. The letter indicates what type of operations occur in the case: I=insert, D=delete, U=update, M=move, C=copy, A=any mix of operations (advanced case) and X=conflict.

Case	**Base**	**Branch 1**	**Branch 2**	**Merge**
I1	<R> <a> <c /> <d /> <e /> </R>	<R> <a> <i1 /> <c /> <d /> <e /> </R>	<R> <a> <c /> <d /> <i2 /> <e /> </R>	<R> <a> <i1 /> <c /> <d /> <i2 /> <e /> </R>
I2	<R> <a /> </R>	<R> <a /> <i1 /> </R>	<R> <a /> <i2 /> </R>	<R> <a /> <i1 /> <i2 /> </R>
	Ambigous merge. Could cause conflict or insert nodes in the order i_2, i_1 instead.			
I3	<R> <a /> <c /> </R>	<R> <a /> <i1 /> <c /> </R>	<R> <a /> <c /> <i2 /> </R>	<R> <a /> <i1 /> <c /> <i2 /> </R>

Case	**Base**	**Branch 1**	**Branch 2**	**Merge**
D1	<R> <s1> <p1 /> <p2 /> </s1> <s2> <p3 /> <p4 /> </s2> </R>	<R> <s1> <p1 /> <p2 /> </s1> </R>	<R> <s1> <p1 /> </s1> <s2> <p3 /> <p4 /> </s2> </R>	<R> <s1> <p1 /> </s1> </R>
D2	<R> <a /> </R>	<R> <a /> </R>	<R> <a /> </R>	<R> <a /> </R>
D3	<R> <a /> <c /> <d /> </R>	<R> <a /> <d /> </R>	<R> <c /> <d /> </R>	<R> <d /> </R>
D4	<R> <a> <c /> </R>	<R> <a> <i> <c /> </i> </R>	<R> <a> <c /> </R>	<R> <a> <i> <c /> </i> </R>
U1	<R> <a /> </R>	<R> <a1 /> </R>	<R> <a /> <b1 /> </R>	<R> <a1 /> <b1 /> </R>
U2	<R> <a /> </R>	<R> <a /> <b1 /> </R>	<R> <a /> <b1 /> </R>	<R> <a /> <b1 /> </R>
U3	<R> <a /> </R>	<R> <a1 /> </R>	<R> <a2 /> </R>	<R> <a1 /> </R>
	Ambigous merge. This file reflects a policy of taking the update from branch 1 when updates conflict.			

<table>
<tr><th>Case</th><th>Base</th><th>Branch 1</th><th>Branch 2</th><th>Merge</th></tr>
<tr><td>M1</td>
<td><R>
  <a>
    <c />
    <d />

    <e>
      <f />
    </e>

</R></td>
<td><R>
  <a>
    <d />
    <c />

    <e>
      <f />
    </e>

</R></td>
<td><R>
  <a>
    <c />
    <d />

    <f>
      <e />
    </f>

</R></td>
<td><R>
  <a>
    <d />
    <c />

    <f>
      <e />
    </f>

</R></td></tr>
<tr><td>M2</td>
<td><R>
  <p1 />
  <p2 />
  <p3 />
  <p4 />
  <p5 />
</R></td>
<td><R>
  <p2 />
  <p1 />
  <p3 />
  <p4 />
  <p5 />
</R></td>
<td><R>
  <p1 />
  <p2 />
  <p3 />
  <p5 />
  <p4 />
</R></td>
<td><R>
  <p2 />
  <p1 />
  <p3 />
  <p5 />
  <p4 />
</R></td></tr>
<tr><td>M3</td>
<td><R>
  <a>
    <a1 />
    <a2 />

    <b1 />
    <b2 />

</R></td>
<td><R>
  <a>
    <a2 />
    <a1 />

    <b1 />
    <b2 />

</R></td>
<td><R>

    <b1 />
    <b2 />

  <a>
    <a1 />
    <a2 />

</R></td>
<td><R>

    <b1 />
    <b2 />

  <a>
    <a2 />
    <a1 />

</R></td></tr>
<tr><td rowspan="2">M4</td>
<td><R>
  <a />

  <c />
</R></td>
<td><R>

    <a />

  <c />
</R></td>
<td><R>

  <c />
  <a />
</R></td>
<td><R>

    <a />

  <c />
</R></td></tr>
<tr><td colspan="4">Moves node a to two different places. The merge shown is the result if the near-move is ignored.</td></tr>
<tr><td>M5</td>
<td><R>
  <a />
  <b />
  <c />
  <d />
  <e />
</R></td>
<td><R>
  <b />
  <c />
  <d />
  <e />
  <a />
</R></td>
<td><R>
  <a />
  <b />
  <d />
  <c />
  <e />
</R></td>
<td><R>
  <b />
  <d />
  <c />
  <e />
  <a />
</R></td></tr>
</table>

Case	Base	Branch 1	Branch 2	Merge
C1	<R> <a /> <b /> <c /> </R>	<R> <a> <b /> <b /> <b /> <c /> </R>	<R> <a /> <b /> <c> <b /> <b /> </c> </R>	<R> <a> <b /> <b /> <b /> <c> <b /> <b /> </c> </R>
C2	<R> <a /> <b /> <c /> <d /> </R>	<R> <a /> <a /> <c /> <d /> </R>	<R> <a /> <b /> <c /> <c /> </R>	<R> <a /> <a /> <c /> <c /> </R>
C3	<R> <a /> <b /> <c /> </R>	<R> <a /> <a /> <b /> <c /> </R>	<R> <a /> <b /> <c /> <c /> </R>	<R> <a /> <a /> <b /> <c /> <c /> </R>
C4	<R> <a /> <b /> <c /> </R>	<R> <a> <b /> <b /> <c /> </R>	<R> <a> <c /> <b /> <c /> </R>	<R> <a> <b /> <c /> <b /> <c /> </R>
	Two nodes copied to same destination. The merged file reflects the "sequence inserts"-policy.			
C5	<R> <a /> <b /> <c /> </R>	<R> <a /> <b /> <a /> <c /> </R>	<R> <a /> <b /> <c /> <a /> </R>	<R> <a /> <b /> <a /> <c /> <a /> </R>

Case	Base	Branch 1	Branch 2	Merge

A1	`<R>`
	`  <a />`
	`  <b />`
	`</R>`
		`<R>`
		`  <a />`
		`  <b1 />`
		`</R>`
			`<R>`
			`  <a />`
			`  <b />`
			`  <c />`
			`</R>`
				`<R>`
				`  <a />`
				`  <b1 />`
				`  <c />`
				`</R>`

A2	`<R>`
	`  <a />`
	`  <b />`
	`  <c />`
	`</R>`
		`<R>`
		`  <a />`
		`  <c />`
		`</R>`
			`<R>`
			`  <a1 />`
			`  <b />`
			`  <c />`
			`</R>`
				`<R>`
				`  <a1 />`
				`  <c />`
				`</R>`

A3	`<R>`
	`  <s1>`
	`    <p1 />`
	`  </s1>`
	`  <s2>`
	`    <p2 />`
	`  </s2>`
	`  <s3>`
	`    <p3 />`
	`  </s3>`
	`</R>`
		`<R>`
		`  <s1>`
		`    <p1 />`
		`  </s1>`
		`  <s2>`
		`    <p2 />`
		`  </s2>`
		`  <s3>`
		`    <p3 />`
		`  </s3>`
		`  <s1>`
		`    <p1 />`
		`  </s1>`
		`</R>`
			`<R>`
			`  <s2>`
			`    <p2 />`
			`  </s2>`
			`  <s1>`
			`    <p1 />`
			`  </s1>`
			`  <s3>`
			`    <p3 />`
			`  </s3>`
			`</R>`
				`<R>`
				`  <s2>`
				`    <p2 />`
				`  </s2>`
				`  <s1>`
				`    <p1 />`
				`  </s1>`
				`  <s3>`
				`    <p3 />`
				`  </s3>`
				`  <s1>`
				`    <p1 />`
				`  </s1>`
				`</R>`

	Conflict due to move of src-of-copy			

A4	`<R>`
	`  <a>`
	`    <b />`
	`    <c />`
	`  </a>`
	`  <d>`
	`    <e />`
	`    <f />`
	`  </d>`
	`</R>`
		`<R>`
		`  <a>`
		`    <c />`
		`    <b />`
		`  </a>`
		`  <d>`
		`    <e />`
		`    <f />`
		`  </d>`
		`</R>`
			`<R>`
			`  <a>`
			`    <b />`
			`    <c />`
			`  </a>`
			`  <a>`
			`    <d>`
			`      <e />`
			`      <f />`
			`    </d>`
			`  </a>`
			`</R>`
				`<R>`
				`  <a>`
				`    <c />`
				`    <b />`
				`  </a>`
				`  <a>`
				`    <d>`
				`      <e />`
				`      <f />`
				`    </d>`
				`  </a>`
				`</R>`

A5	`<R>`
	`  <a />`
	`  <b />`
	`</R>`
		`<R>`
		`  <a1 />`
		`  <b />`
		`</R>`
			`<R>`
			`  <a />`
			`  <b>`
			`    <a />`
			`    <b />`
			`  </b>`
			`</R>`
				`<R>`
				`  <a1 />`
				`  <b>`
				`    <a1 />`
				`    <b />`
				`  </b>`
				`</R>`

Case	**Base**	**Branch 1**	**Branch 2**	**Merge**

A6	`<R>`
	`<a>`
	`<b />`
	`<c />`
	`</a>`
	`</R>`
		`<R>`
		`<a>`
		`<c />`
		`<b />`
		`</a>`
		`</R>`
			`<R>`
			`<a>`
			`<b />`
			`<c />`
			`</a>`
			`<a>`
			`<b />`
			`<c />`
			`</a>`
			`</R>`
				`<R>`
				`<a>`
				`<c />`
				`<b />`
				`</a>`
				`<a>`
				`<c />`
				`<b />`
				`</a>`
				`</R>`

A7	`<R>`
	`<a />`
	`<b />`
	`<c />`
	`</R>`
		`<R>`
		`<a />`
		`<a />`
		`<c />`
		`</R>`
			`<R>`
			`<a />`
			`<c />`
			`<b />`
			`</R>`
				`<R>`
				`<a />`
				`<a />`
				`<c />`
				`</R>`

	The node b is moved and deleted. The current implementation deletes the node and ignores the move.			

A8	`<R>`
	`<a>`
	`<b />`
	`<c />`
	`</a>`
	`</R>`
		`<R>`
		`<a>`
		`<b />`
		`</a>`
		`<c />`
		`</R>`
			`<R>`
			`<a>`
			`<b />`
			`<c />`
			`</a>`
			`<a>`
			`<b />`
			`<c />`
			`</a>`
			`</R>`
				`<R>`
				`<a>`
				`<b />`
				`</a>`
				`<a>`
				`<b />`
				`</a>`
				`<c />`
				`</R>`

	Not enough context, should cause hangon sequence conflict			

A9	`<R>`
	`<a>`
	`<b />`
	`<c />`
	`</a>`
	`</R>`
		`<R>`
		`<a>`
		`<b />`
		`<c />`
		`</a>`
		`<a>`
		`<c />`
		`<b />`
		`</a>`
		`</R>`
			`<R>`
			`<a>`
			`<b />`
			`<c />`
			`<d />`
			`</a>`
			`</R>`
				`<R>`
				`<a>`
				`<b />`
				`<c />`
				`<d />`
				`</a>`
				`<a>`
				`<c />`
				`<b />`
				`</a>`
				`</R>`

Case	Base	Branch 1	Branch 2	Merge
A10	<R> <l> <a /> </l> <l> </l> </R>	<R> <l> </l> <l> <a /> </l> </R>	<R> <l> </l> <l> </l> </R>	<R> <l> </l> <l> </l> </R>
	Fails because matchings aren't generated as assumed in the merge, i.e. that the two subtrees rooted at l are swapped.			
A11	<R> <a /> <c /> </R>	<R> <a /> <c /> </R>	<R> <a /> <c /> <i /> </R>	<R> <a /> <c /> <i /> </R>
A12	<R> <a /> </R>	<R> <a /> <a /> </R>	<R> <a /> <i /> </R>	<R> <a /> <a /> <i /> </R>
A13	<R> <p1 /> <p2 /> </R>	<R> <p1 /> <p2P /> </R>	<R> <p1 /> <a> <p2 /> </R>	<R> <p1 /> <a> <p2P /> </R>
A14	<R> <a /> </R>	<R> <aP /> <bP /> </R>	<R> <a /> </R>	<R> <aP /> <bP /> <bP /> </R>
A15	<R> <a> <d /> <e /> <f /> </R>	<R> <f /> <a> <d /> <e /> <i /> </R>	<R> <a> <e /> <d /> <bP /> </R>	<R> <bP /> <a> <e /> <d /> <i /> </R>

Case	Base	Branch 1	Branch 2	Merge
A16	<R> <a> <m /> <c> <cc /> </c> </R>	<Rp> <a> <m> <i /> </m> <c> <cc /> </c> </Rp>	<R> <c> <cc /> </c> <c> <cc /> </c> <m /> </R>	<Rp> <c> <cc /> </c> <c> <cc /> </c> <m> <i /> </m> </Rp>
X1	<R> <a /> </R>	<R> <a /> <b2 /> </R>	<R> <a /> <b1 /> </R>	<R> <a /> <b2 /> </R>
	Update/update conflict. The node b is updated to b_1 and b_2.			
X2	<R> <a /> </R>	<R> <a /> <a /> </R>	<R> </R>	<R> <a /> </R>
	Source of copy is deleted. Merge shows result if the delte is ignored.			
X3	<R> <a /> </R>	<R> <a /> <a /> </R>	<R> <a /> </R>	<R> <a /> <a /> </R>
	There is no conflict here. The case is mislabeled.			
X4	<R> <a /> <c /> </R>	<R> <a /> <c /> </R>	<R> <a /> <c /> </R>	<R> <a /> <c /> </R>
	Sequencing conflict. The merge shows the sequence of branch 1.			
X5	<R> <a /> <c> <d> <f /> <g /> </d> <e /> </c> </R>	<R> <a /> <c> <d> <g /> <f /> </d> <eP /> </c> </R>	<R> <d> <f /> <g /> </d> <a /> </R>	<R> <d> <g /> <f /> </d> <a /> </R>

Case	**Base**	**Branch 1**	**Branch 2**	**Merge**
	Update/delete conflict. The node e is updated in branch 1 and a subtree containing the node is deleted in branch 2.			
X6	<R> <a> <c> <d /> </c> </R>	<R> <a> <c> <d /> </c> </R>	<R> <d> <c> <a /> </c> </d> </R>	
	Puts merging algorithm in infinite loop with inappropriate matching.			

Appendix H

3DM Source Code

Contents

BaseNode.java 173
BranchNode.java 173
ConflictLog.java 174
Diff.java 175
DiffMatching.java 176
EditLog.java 176
MatchArea.java 177
MatchedNodes.java 177
Matching.java 178
Measure.java 181
Merge.java 182
Node.java 187
NodeFactory.java 188
ParseException.java 188
Patch.java 188
PathTracker.java 189
TreeDiffMerge.java 190
TriMatching.java 191
XMLElementNode.java 192
XMLNode.java 192
XMLParser.java 193
XMLPrinter.java 193
XMLTextNode.java 194

BaseNode.java **Page 1 of 1**

```
// $Id: BaseNode.java,v 1.7 2001/09/05 13:21:24 ctl Exp $ D

/**
 * Node in a base tree matched witch is matched to two branches. In addition to
 * the functionality provided by the node class, BaseNode adds matchings. Each
 * BaseNode can be matched to multipleBranchNodes. Matches to the node in th
e
 * left and right branches are accesed with the {@link #getLeft() getLeft} and
 * {@link #getRight() getRight} methods.
 */

public class BaseNode extends Node {

  // Left and right matches
  private MatchedNodes left=null;
  private MatchedNodes right=null;

  public BaseNode( XMLNode aContent ) {
    super();
    left = new MatchedNodes(this);
    right = new MatchedNodes(this);
    content = aContent;
  }

  public BaseNode getChild( int ix ) {
    return (BaseNode) children.elementAt(ix);
  }

  public BaseNode getParent() {
    return (BaseNode) parent;
  }

  public MatchedNodes getLeft() {
    return left;
  }

  public MatchedNodes getRight() {
    return right;
  }

  public void swapLeftRightMatchings() {
    MatchedNodes t = left;
    left=right;
    right=t;
  }

}
```

BranchNode.java **Page 1 of 2**

```
// $Id: BranchNode.java,v 1.10 2001/09/05 13:21:25 ctl Exp $ D

import java.util.Iterator;

/**
 * Node in a branch tree. In addition to the
 * functionality provided by the node class, BranchNode adds matchings to a
 * node in the base tree.
 */

public class BranchNode extends Node {

  // Match types
  public static final int MATCH_FULL = 3;
  public static final int MATCH_CONTENT = 1;
  public static final int MATCH_CHILDREN = 2;

  private MatchedNodes partners = null;
  private BaseNode baseMatch = null;
  private int matchType = 0;

  public BranchNode( XMLNode aContent ) {
    super();
    content = aContent;
  }

  public BranchNode getChild( int ix ) {
    return (BranchNode) children.elementAt(ix);
  }

  public BranchNode getParent() {
    return (BranchNode) parent;
  }

  public void setPartners(MatchedNodes p) {
    partners = p;
  }

  public MatchedNodes getPartners() {
    return partners;
  }

  public void setBaseMatch(BaseNode p, int amatchType) {
    if( amatchType < MATCH_CONTENT || amatchType > MATCH_FULL )
      throw new IllegalArgumentException();
    baseMatch = p;
    matchType = amatchType;
  }

  public void setMatchType( int amatchType ) {
    if( amatchType < MATCH_CONTENT || amatchType > MATCH_FULL )
      throw new IllegalArgumentException();
    matchType = amatchType;
  }

  public void delBaseMatch() {
    baseMatch = null;
    matchType = 0;
  }

  public int getBaseMatchType() {
    return matchType;
  }

  public boolean hasBaseMatch() {
    return baseMatch != null;
  }
```

BranchNode.java **Page 2 of 2**

```
  public BaseNode getBaseMatch() {
    return baseMatch;
  }

  /** Tells if this node is in the left tree. */
  public boolean isLeftTree() {
    // Assumes that at least the root is matched.
    if( baseMatch != null )
      return baseMatch.getLeft().getMatches().contains(this);
    else
      return getParent().isLeftTree();
  }

  public boolean isMatch( int type) {
    return ((matchType & type) != 0);
  }


  /** Find a node partner of given type. */

  public BranchNode getFirstPartner( int typeFlags ) {
    // Remeber to check both steps! The canidate's match type is only from base
    // if A should match B structurally we need
    // A---------Base---------B
    //    struct      struct
    if( ( matchType & typeFlags) == 0 )
      return null;
    MatchedNodes m= getPartners();
    if( m == null )
      return null;
    for( Iterator i = m.getMatches().iterator();i.hasNext();) {
      BranchNode candidate = (BranchNode) i.next();
      if((candidate.matchType & typeFlags) != 0)
        return candidate;
    }
    return null;
  }
}
```

ConflictLog.java **Page 1 of 3**

```java
// $Id: ConflictLog.java,v 1.6 2001/09/05 13:21:25 ctl Exp $ D

import java.util.List;
import java.util.LinkedList;
import java.util.Iterator;
import org.xml.sax.ContentHandler;
import org.xml.sax.helpers.AttributesImpl;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;

/** Class for logging conflicts and conflict warnings.
 */
public class ConflictLog {

  // Types of conflicts (used in add... functions)
  public static final int UPDATE = 1;
  public static final int DELETE = 2;
  public static final int INSERT = 3;
  public static final int MOVE = 4;

  private final String[] TYPETAGS = {null,"update","delete","insert","move"};
  private LinkedList conflicts = new LinkedList();
  private LinkedList warnings = new LinkedList();
  private PathTracker pt = null;

  public ConflictLog(PathTracker apt) {
    pt=apt;
  }

  private class ConflictEntry {
    String text = null;
    BaseNode b=null;
    BranchNode b1=null,b2=null;
    String mergePath = null;
    int type = 0;
  }

  public void addListConflict( int type, String text, BaseNode b, BranchNode ba,
BranchNode bb ) {
    add( true, false, type,text,b,ba,bb);
  }

  public void addListWarning( int type, String text, BaseNode b, BranchNode ba,
                              BranchNode bb ) {
    add( true, true, type,text,b,ba,bb);
  }

  public void addNodeConflict( int type, String text, BaseNode b, BranchNode b
a,
                              BranchNode bb ) {
    add( false, false,type, text,b,ba,bb);
  }

  public void addNodeWarning( int type, String text, BaseNode b, BranchNode b
a,
                              BranchNode bb ) {
    add( false, true, type,text,b,ba,bb);
  }

  protected void add( boolean list, boolean warning, int type, String text,
                      BaseNode b, BranchNode ba, BranchNode bb ) {
    ConflictEntry ce = new ConflictEntry();
    ce.text = text;
    ce.type = type;
    ce.b = b;
    // let b1=left and b2=right
    if( ba == null ) {
```

ConflictLog.java **Page 2 of 3**

```java
      ba=bb;
      bb=null;
    }
    if( ba != null && ba.isLeftTree() ) {
      ce.b1 = ba;
      ce.b2 = bb;
    } else {
      ce.b1 = bb;
      ce.b2 = ba;
    }
    if( list ) {
      ce.mergePath=pt.getPathString();
    } else
      ce.mergePath=pt.getFullPathString();
    if( warning )
      warnings.addLast(ce);
    else
      conflicts.addLast(ce);
  }

  /** Output conflict list as XML.
   * @param ch Content handler to output the conflict to.
   */
  public void writeConflicts( ContentHandler ch ) throws SAXException {
    AttributesImpl atts = new AttributesImpl();
    ch.startDocument();
    ch.startElement("","","conflictlist",atts);
    if( !conflicts.isEmpty() ) {
      atts = new AttributesImpl();
      ch.startElement("","","conflicts",atts);
      for( Iterator i = conflicts.iterator();i.hasNext();)
        outputConflict( (ConflictEntry) i.next(),ch );
      ch.endElement("","","conflicts");
    }
    if( !warnings.isEmpty() ) {
      atts = new AttributesImpl();
      ch.startElement("","","warnings",atts);
      for( Iterator i = warnings.iterator();i.hasNext();)
        outputConflict( (ConflictEntry) i.next(),ch );
      ch.endElement("","","warnings");
    }
    ch.endElement("","","conflictlist");
    if( !conflicts.isEmpty() )
      System.out.println( "MERGE FAILED: " + conflicts.size() + " conflicts.");
    if( !warnings.isEmpty() )
      System.out.println( "Warning: " + warnings.size() +" conflict warnings.");
    ch.endDocument();
  }

  protected void outputConflict( ConflictEntry ce, ContentHandler ch )
    throws SAXException {
    AttributesImpl atts = new AttributesImpl();
    ch.startElement("","",TYPETAGS[ce.type],atts);
    ch.characters(ce.text.toCharArray(),0,ce.text.length());
    atts = new AttributesImpl();
    atts.addAttribute("","","tree","CDATA","merged");
    atts.addAttribute("","","path","CDATA",ce.mergePath);
    ch.startElement("","","node",atts);
    ch.endElement("","","node");

    if( ce.b!= null ) {
      atts = new AttributesImpl();
      atts.addAttribute("","","tree","CDATA","base");
      atts.addAttribute("","","path","CDATA",PathTracker.getPathString(ce.b));
      ch.startElement("","","node",atts);
      ch.endElement("","","node");
    }
```

ConflictLog.java **Page 3 of 3**

```java
    if( ce.b1!= null ) {
      atts = new AttributesImpl();
      atts.addAttribute("","","tree","CDATA","branch1");
      atts.addAttribute("","","path","CDATA",PathTracker.getPathString(ce.b1));
      ch.startElement("","","node",atts);
      ch.endElement("","","node");
    }
    if( ce.b2!= null ) {
      atts = new AttributesImpl();
      atts.addAttribute("","","tree","CDATA","branch2");
      atts.addAttribute("","","path","CDATA",PathTracker.getPathString(ce.b2));
      ch.startElement("","","node",atts);
      ch.endElement("","","node");
    }
    ch.endElement("","",TYPETAGS[ce.type]);
  }
}
```

```java
// $Id: Diff.java,v 1.4 2001/09/05 13:21:25 ctl Exp $ D

import org.xml.sax.ContentHandler;
import org.xml.sax.helpers.AttributesImpl;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import java.util.Vector;
import java.util.Set;
import java.util.HashSet;
import java.util.Map;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Iterator;

/** Produces the diff between two naturally matched trees.
 *  Collapsing multiple copy-ops using the run attribute is not implemented in
 *  this version.
 */

public class Diff {

  private Matching m = null;
  private static final Attributes EMPTY_ATTS = new AttributesImpl();
  private static final Set RESERVED;
  private static final String DIFF_NS ="diff:";
  static {
      RESERVED = new HashSet();
      RESERVED.add(DIFF_NS+"copy");
      RESERVED.add(DIFF_NS+"insert");
      RESERVED.add(DIFF_NS+"esc");
  }

  /** Construct a diff operating on the matched trees passed to the constructor.
   *  Note that the matching contains pointers to the base and new trees.
   *  @param am Matching between trees to diff
   */
  public Diff(Matching am) {
    m=am;
  }

  /** Encode the diff between the trees passed to the constructor.
   *  @param ch Output encoder for the diff
   */
  public void diff( ContentHandler ch ) throws SAXException {
    enumerateNodes(m.getBaseRoot(),nodeNumbers);
    ch.startDocument();
    ch.startElement("","","diff",EMPTY_ATTS);
    copy( m.getBaseRoot(), m.getBranchRoot(), ch );
    ch.endElement("","","diff");
    ch.endDocument();
  }

  protected void copy( BaseNode b, BranchNode branch, ContentHandler ch )
                     throws SAXException {
    // Find stopnodes
    Vector stopNodes = new Vector();
    m.getAreaStopNodes( stopNodes, branch );
    for( Iterator i = stopNodes.iterator();i.hasNext();) {
      BranchNode stopNode = (BranchNode) i.next();
      String dst = getId(stopNode.getBaseMatch());
      for( int ic = 0;ic < stopNode.getChildCount();ic++) {
        BranchNode child = stopNode.getChild(ic);
        if( child.hasBaseMatch() ) {
          // Copy tree...
          AttributesImpl copyAtts = new AttributesImpl();
          copyAtts.addAttribute("","","src","CDATA",getId(child.getBaseMatch()));
          copyAtts.addAttribute("","","dst","CDATA",dst);
```



```java
          ch.startElement("","",DIFF_NS+"copy",copyAtts);
          copy( child.getBaseMatch(), child, ch );
          ch.endElement("","",DIFF_NS+"copy");
        } else {
          // Insert tree...
          AttributesImpl copyAtts = new AttributesImpl();
          copyAtts.addAttribute("","","dst","CDATA",dst);
          ch.startElement("","",DIFF_NS+"insert",copyAtts);
          insert( child, ch );
          ch.endElement("","",DIFF_NS+"insert");
        }
      } // endfor children
    }
  }

  protected void insert( BranchNode branch, ContentHandler ch )
                     throws SAXException {
    XMLNode content = branch.getContent();
    if( content instanceof XMLTextNode ) {
      XMLTextNode ct = (XMLTextNode) content;
      ch.characters(ct.getText(),0,ct.getText().length);
    } else {
      // Element node
      XMLElementNode ce = (XMLElementNode) content;
      boolean escape = RESERVED.contains(ce.getQName());
      if( escape ) ch.startElement("","",DIFF_NS+"esc",EMPTY_ATTS);
      ch.startElement(ce.getNamespaceURI(),ce.getLocalName(),ce.getQName(),
ce.getAttributes());
      if( escape ) ch.endElement("","",DIFF_NS+"esc");
      for( int i=0;i<branch.getChildCount();i++) {
        BranchNode child = branch.getChild(i);
        if( child.hasBaseMatch() ) {
          AttributesImpl copyAtts = new AttributesImpl();
          copyAtts.addAttribute("","","src","CDATA",
            getId(child.getBaseMatch()));
          ch.startElement("","",DIFF_NS+"copy",copyAtts);
          copy( child.getBaseMatch(), child, ch );
          ch.endElement("","",DIFF_NS+"copy");
        } else
          insert( child, ch );
      }
      if( escape ) ch.startElement("","",DIFF_NS+"esc",EMPTY_ATTS);
      ch.endElement(ce.getNamespaceURI(),ce.getLocalName(),ce.getQName());
      if( escape ) ch.endElement("","",DIFF_NS+"esc");
    }
  }

  protected Map nodeNumbers = new HashMap();

  // Get BFS number of node
  protected String getId( Node n ) {
    return ((Integer) nodeNumbers.get(n)).toString();
  }

  // BFS Enumeration of nodes. Useful beacuse adjacent nodes have subsequent
ids
  // => diff can use the "run" attribute more often
  protected void enumerateNodes( Node start, Map map ) {
    int id = 0;
    LinkedList queue = new LinkedList();
    queue.add(start);
    while( !queue.isEmpty() ) {
      Node n = (Node) queue.removeFirst();
      map.put(n,new Integer(id));
      for( int i=0;i<n.getChildCount();i++)
        queue.add(n.getChildAsNode(i));
      id++;
```



```java
    }
  }
}
```

DiffMatching.java Page 1 of 1

```java
// $Id: DiffMatching.java,v 1.3 2001/09/05 13:21:25 ctl Exp $ D

import Matching;
import java.util.Vector;
import java.util.Iterator;

/** Tree matching suitable for producing diffs. Compared to the standard
 *  matching, DiffMatching does not match nodes with similar content (there's
 *  no point to that), and tries to find matches that form uninterrupted runs
 *  of src attributes => more efficient encoding of the diff.
 */

public class DiffMatching extends Matching {

  /** Construct a matching between a base and branch tree. */
  public DiffMatching(BaseNode abase, BranchNode abranch ) {
    super( abase, abranch );
  }

  protected void buildMatching( BaseNode base, BranchNode branch ) {
    matchSubtrees( base, branch );
  }

  // We never match fuzzy when diffing
  protected boolean dfsTryFuzzyMatch( Node a, Node b) {
    return false;
  }

  // Only find exact candidates
  protected Vector findCandidates( BaseNode tree, BranchNode key ) {
    Vector candidates = new Vector();
    findExactMatches( tree, key, candidates );
    return candidates;
  }

  protected CandidateEntry getBestCandidate(  BranchNode branch,
                                        Vector bestCandidates ) {
    // Try to return a node who is next to the previously matched node
    // (sequencing of src nodes!)
    if( bestCandidates.size() > 1 ) {
      for( Iterator i = bestCandidates.iterator();i.hasNext();) {
        CandidateEntry ce = (CandidateEntry) i.next();
        BranchNode left = (BranchNode) branch.getLeftSibling();
        BaseNode cand = ce.candidate;
        if( left != null && left.hasBaseMatch() && left.getBaseMatch() ==
            cand.getLeftSibling() )
          return ce;
      }
    }
    if( bestCandidates.isEmpty() )
      return null;
    else
      return (CandidateEntry) bestCandidates.elementAt(0);
  }
}
```

EditLog.java Page 1 of 3

```java
// $Id: EditLog.java,v 1.4 2001/09/05 13:21:25 ctl Exp $ D

import java.util.Vector;
import java.util.Stack;

import org.xml.sax.SAXException;
import org.xml.sax.ContentHandler;
import org.xml.sax.helpers.AttributesImpl;

/** Logs edit operations perfomed by the merge algorithm. */

public class EditLog {

  // Inyternal edit op codes
  private static final int INSERT = 0;
  private static final int UPDATE = 1;
  private static final int COPY = 2;
  private static final int MOVE = 3;
  private static final int DELETE = 4;

  // Edit tag names
  private static final String[] OPTAGS = {"insert","update","copy","move",
                                         "delete"};

  // Class for storing an edit operation in memory.
  private class EditEntry {
    int type = -1;
    BaseNode baseSrc=null;
    BranchNode branchSrc = null;
    String dstPath = null;

    EditEntry( int aType, BaseNode aBaseNode, BranchNode aBranchNode,
            String aDstPath ) {
      type = aType;
      baseSrc = aBaseNode;
      branchSrc = aBranchNode;
      dstPath = aDstPath;
    }
  }

  private Stack checkPoints = new Stack();
  // Edits in the log. A list of EditEntries.
  private Vector edits = new Vector();
  private PathTracker pt = null;

  /** Construct edit log. The PathTracker given as argument is queried for the
   *  current position in the merge treeeach time each time an edit operation is
   *  added.
   *  @param apt PathTracker that tracks the current position in the merged tree
   */
  public EditLog(PathTracker apt) {
    pt = apt;
  }

  /** Add insert operation.
   *  @param n Node that is inserted
   *  @param childPos position in the current child list of the merge tree */
  public void insert( BranchNode n, int childPos ) {
    edits.add( new EditEntry(INSERT,null,n,pt.getPathString(childPos)));
  }

  /** Add move operation.
   *  @param n Node that is moved
   *  @param childPos position in the current child list of the merge tree */
  public void move( BranchNode n, int childPos ) {
    edits.add( new EditEntry(MOVE,n.getBaseMatch(),n,
                       pt.getPathString(childPos)));
```

EditLog.java Page 2 of 3

```java
  }

  /** Add copy operation.
   *  @param n Node that is copied
   *  @param childPos position in the current child list of the merge tree */
  public void copy( BranchNode n, int childPos ) {
    edits.add( new EditEntry(COPY,n.getBaseMatch(),n,
                       pt.getPathString(childPos)));
  }

  /** Add move operation.
   *  @param n Node that is upated.
   *  @param childPos position in the current child list of the merge tree */
  public void update( BranchNode n ) {
    edits.add( new EditEntry(UPDATE,n.getBaseMatch(),n,pt.getFullPathString()))
;
  }

  /** Add delete operation.
   *  @param n Node that is deleted
   *  @param originatingList Node, whose child list originated the delete.
   */
  public void delete( BaseNode n, BranchNode originatingList ) {
    edits.add( new EditEntry(DELETE,n,originatingList,null));
  }

  /** Write out the edit log. */
  public void writeEdits( ContentHandler ch ) throws SAXException {
    ch.startDocument();
    AttributesImpl atts = new AttributesImpl();
    ch.startElement("","","edits",atts);
    for(int i=0;i<edits.size();i++)
      outputEdit((EditEntry) edits.elementAt(i),ch);
    ch.endElement("","","edits");
    ch.endDocument();
  }

  protected void outputEdit( EditEntry ee, ContentHandler ch )
                       throws SAXException {
    AttributesImpl atts = new AttributesImpl();
    if( ee.type != DELETE )
      atts.addAttribute("","","path","CDATA",ee.dstPath);
    if( ee.type != INSERT )
      atts.addAttribute("","","src","CDATA",PathTracker.getPathString(ee.baseSr
c));
    atts.addAttribute("","","originTree","CDATA",ee.branchSrc.isLeftTree() ?
                "branch1" : "branch2");
    atts.addAttribute("","",(ee.type != DELETE ) ? "originNode" : "originList",
      "CDATA",PathTracker.getPathString(ee.branchSrc));
    ch.startElement("","",OPTAGS[ee.type],atts);
    ch.endElement("","",OPTAGS[ee.type]);
  }

  /** Mark a checkpoint in the edit log. */
  public void checkPoint() {
    checkPoints.push(new  Integer( edits.size() ) );
  }

  /** Remove all edits added after the last checkpoint. */
  public void rewind() {
    int firstFree = ((Integer) checkPoints.pop()).intValue();
    edits.setSize(firstFree);
  }

  /** Commit edits made after the last checkpoint. */
  public void commit() {
    checkPoints.pop();
```

EditLog.java **Page 3 of 3**

```
  }
}
```

MatchArea.java **Page 1 of 1**

```
//$Id: MatchArea.java,v 1.3 2001/09/05 13:21:26 ctl Exp $ D

/** Class used to tag nodes in the same matched subtree. The class also
 *  contains fields for the root and information size of the subtree. */

public class MatchArea {

  private int infoBytes = 0;
  private BranchNode root = null;

  public MatchArea( BranchNode aRoot ) {
    root = aRoot;
  }

  public BranchNode getRoot() {
    return root;
  }

  public void addInfoBytes( int i) {
    infoBytes+=i;
  }

  public int getInfoBytes() {
    return infoBytes;
  }
}
```

MatchedNodes.java **Page 1 of 1**

```
// $Id: MatchedNodes.java,v 1.6 2001/09/05 13:21:26 ctl Exp $ D

import java.util.Set;
import java.util.HashSet;
import java.util.Iterator;

/** Container for a set of nodes, matched to the node owning the container. */

public class MatchedNodes {

  private BaseNode owner=null;
  private Set matches=new HashSet();

  /** Create a new conatiner of matched nodes. All nodes in the container are
   *  matched to the owner node.
   *  @param aowner Owner of the container.
   */
  public MatchedNodes(BaseNode aowner) {
    owner = aowner;
  }

  public void addMatch(BranchNode n) {
    matches.add(n);
  }

  public void delMatch(BranchNode n) {
    matches.remove(n);
  }

  public Set getMatches() {
    return matches;
  }

  public int getMatchCount() {
    return matches.size();
  }

  /** Get the first node that is fully matched to the owner. */

  public BranchNode getFullMatch() {
    for( Iterator i=matches.iterator();i.hasNext();) {
      BranchNode fmatch = (BranchNode) i.next();
      if( fmatch.isMatch(BranchNode.MATCH_FULL))
        return fmatch;
    }
    return null;
  }
}
```

```
// $Id: Matching.java,v 1.19 2001/09/05 13:21:26 ctl Exp $ D

import java.util.Vector;
import java.util.Iterator;
import java.util.SortedSet;
import java.util.TreeSet;
import java.util.HashSet;
import java.util.Set;
import java.util.Comparator;
import java.util.Collections;

/** Builds a matching between two trees. The actual matchings are stored in
 *  the nodes of the trees (the BaseNode and BranchNode objects).
 *  Note: make sure there are no matched nodes in the trees before
 *  constructing a matching.
 */

public class Matching {

  protected BaseNode baseRoot = null;
  private BranchNode branchRoot = null;

  protected Matching() {
    // Does nothing, only called from TriMatching constructor
  }

  /** Build matching between two trees.
   *  @param abase  Root of the base tree
   *  @param abranch Root of the branch tree
   */
  public Matching(BaseNode abase, BranchNode abranch ) {
    baseRoot = abase;
    branchRoot = abranch;
    buildMatching( baseRoot, branchRoot );
  }

  protected void buildMatching( BaseNode base, BranchNode branch ) {
    matchSubtrees( base, branch );
    removeSmallCopies(branch);
    matchSimilarUnmatched( base, branch );
    setMatchTypes(base);
    // Artificial roots always matched (otherwise BranchNode.isLeft() may fail!)
    branch.setBaseMatch(base,BranchNode.MATCH_FULL);
  }

  public BaseNode getBaseRoot() {
    return baseRoot;
  }

  public BranchNode getBranchRoot() {
    return branchRoot;
  }

  protected void matchSubtrees( BaseNode base, BranchNode branch ) {
    // Find candidates for node branch in base
    Vector candidates = findCandidates( base, branch );
    // Find best matching trees
    int bestCount = 0;
    CandidateEntry best = null;
    Vector bestCandidates = new Vector();
    for( Iterator i = candidates.iterator(); i.hasNext(); ) {
      CandidateEntry candidate = (CandidateEntry) i.next();
      int initCount = 0;
      int thisCount = dfsMatch( candidate.candidate, branch, initCount );
      if( thisCount == bestCount )
        bestCandidates.add( candidate );
      else if( thisCount > bestCount ) {
```



```
        bestCount = thisCount;
        bestCandidates.clear();
        bestCandidates.add( candidate );
      }
    }
    // Add matchings and find nodes below matching subtree
    best = getBestCandidate(branch,bestCandidates,bestCount);
    Vector stopNodes = new Vector();
    if( best != null )
      dfsMatch( best.candidate, branch, 0, stopNodes, new MatchArea(branch) );
    else {
      // Unmatched
      for( int i=0;i<branch.getChildCount();i++)
        stopNodes.add(branch.getChild(i));
    }
    // Recurse
    for( Iterator i=stopNodes.iterator();i.hasNext();)
      matchSubtrees(base,(BranchNode) i.next());
  }

  /*** Select the best matching candidate from a set of candidate subtrees.
   * @param branch  The node we're trying to match
   * @param bestCandiates vector of potential candidate in the base tree. The
   *                  vector consists of <code>CandidateEntry</code>:s
   * @param bestCount size (in nodes) of largest candidate subtree. */
  protected CandidateEntry getBestCandidate( BranchNode branch,
                           Vector bestCandidates, int bestCount ) {
    CandidateEntry best=null;
    // Resolve ambiguities if any exist...
    if( bestCandidates.size() > 1 ) {
      // Check if left neighbor of candidate is matched to left of base,
      // in that case we have a clear winner!
      for( Iterator i = bestCandidates.iterator(); i.hasNext(); ) {
        CandidateEntry candidate = (CandidateEntry) i.next();
        BranchNode left = (BranchNode) branch.getLeftSibling();
        if( left != null && left.hasBaseMatch() && left.getBaseMatch() ==
          candidate.candidate.getLeftSibling() ) {
          return candidate;
        }
      }
      // Didn't work..now we've try to make a judgement based on context
      // Ambiguities - we need to find out which one is the best...
      // First calc all missing left-right correlations
      for( Iterator i = bestCandidates.iterator(); i.hasNext(); ) {
        CandidateEntry candidate = (CandidateEntry) i.next();
        if( candidate.leftRightDown < 0.0 ) {
          candidate.leftRightDown = Math.min(
            measure.childListDistance(candidate.candidate,branch),
            Math.min( getDistanceOfRight(candidate.candidate,branch),
              getDistanceOfLeft(candidate.candidate,branch)));
        }
      }
      Collections.sort(bestCandidates,candlrdComp);
    }
    best = bestCandidates.isEmpty() ? null :
        (CandidateEntry) bestCandidates.elementAt(0);
    if( best!=null && (bestCount == 1 &&
      (Math.min(best.leftRightDown,best.distance) > 0.1 )))
        best=null;
    return best;
  }

  private void matchSimilarUnmatched( BaseNode base, BranchNode branch) {
    // Traverse in preorder -- to avoid building trees, just fixing levels where
    // parents are matched
    for( int i=0;i<branch.getChildCount();i++)
      matchSimilarUnmatched(base,branch.getChild(i));
```



```
    BaseNode baseMatch = branch.getBaseMatch(); // old  baseparent
    if( baseMatch != null && baseMatch.getChildCount()>0 ) {
      // Scan for unmapped nodes
      for( int i=0;i<branch.getChildCount();i++) {
        BranchNode n = branch.getChild(i);
        BaseNode leftcand=null,rightcand=null;
        int lastBaseChild = baseMatch.getChildCount()-1;
        if( n.getBaseMatch() != null)
          continue; // Mapped, all is well
        // end points
        if( i == 0 && !isMatched(baseMatch.getChild(0)) ){
          addMatchingIfSameType(n, baseMatch.getChild(0) );
          continue;
        } else if (i==branch.getChildCount()-1 && !isMatched(baseMatch.getChild(la
stBaseChild))) {
          addMatchingIfSameType(n, baseMatch.getChild(lastBaseChild) );
          continue;
        }
        if( i > 0 ) {
          // See if node preceding n is matched,and its right neighbor unmatched
          // Base    xy     p=n's predecessor, x matches p and y unmatched
          // Branch  pn       => match y and n
          BaseNode x = branch.getChild(i-1).getBaseMatch();
          if( x != null && x.hasRightSibling() && !isMatched((BaseNode)
                                                x.getRightSibling())) {
            addMatchingIfSameType(n,(BaseNode) x.getRightSibling() );
            continue;
          }
        }
        if( i < branch.getChildCount()-1 ) {
          // See if node succeeding n is matched, and its right neighbor
          // unmatched
          // Base    yx     p=n's succecessor, x matches p and y unmatched
          // Branch  np       => match y and n
          BaseNode x = branch.getChild(i+1).getBaseMatch();
          if( x != null && x.hasLeftSibling() && !isMatched((BaseNode)
                                                  x.getLeftSibling())) {
            addMatchingIfSameType(n,(BaseNode) x.getLeftSibling() );
            continue;
          }
        }
      } // endfor
    } // endif
  }

  private Set removedMatchings = new HashSet();

  private void setMatchTypes( BaseNode base ) {
    // Postorder traversal, because we use the child mappings to orient us
    // therefore, this better change child mappings first!
    for(int i=0;i<base.getChildCount();i++)
      setMatchTypes(base.getChild(i));
    if( base.getLeft().getMatches().size() > 1 ) {
      // Has multiple matches, need to determine type of each copy
      // Scan for primary copy...
      int minDist = Integer.MAX_VALUE;
      double minContentDist = Double.MAX_VALUE;
      BranchNode master = null;
      for( Iterator i=base.getLeft().getMatches().iterator();i.hasNext();) {
        BranchNode cand = (BranchNode) i.next();
        int dist = exactChildListMatch(base,cand) ? 0 :
                measure.matchedChildListDistance( base, cand );
        if( dist < minDist ) {
          minDist = dist;
          master = cand;
        } else if( dist == minDist ) {
```

```
        // minContentDist not neccessarily calced already, do it now.
        minContentDist = measure.getDistance( base, master );
        double cDist = measure.getDistance( base, cand );
        if( cDist < minContentDist ) {
          minContentDist = cDist;
          master = cand;
        }
      }
    }
    removedMatchings.clear();
    // Master is now the best match, which will be assigned as MATCH_FULL
    for( Iterator i=base.getLeft().getMatches().iterator();i.hasNext();) {
      BranchNode cand = (BranchNode) i.next();
      if( cand == master )
        continue;
      boolean structMatch = exactChildListMatch(base,cand);
      boolean contMatch = cand.getContent().contentEquals(base.getContent());
      if( !structMatch && !contMatch )
        removedMatchings.add( cand );
      else
        cand.setMatchType( (contMatch ? BranchNode.MATCH_CONTENT : 0) +
                 (structMatch ? BranchNode.MATCH_CHILDREN : 0) );
    }
    // Delete any removed matchings
    for( Iterator i = removedMatchings.iterator();i.hasNext();) {
      BranchNode cand = (BranchNode) i.next();
      delMatching( cand, cand.getBaseMatch() );
    }
  } // If node copied
}

/** Check if the child lists of two nodes are matched exactly. */
private boolean exactChildListMatch( BaseNode base, BranchNode a) {
  if( a.getChildCount() != base.getChildCount() )
    return false;
  for(int i=0;i<a.getChildCount();i++) {
    if( base.getChild(i) != a.getChild(i).getBaseMatch() )
      return false;
  }
  return true;
}

// Threshold for considering a subtree to be copied
public static int COPY_THRESHOLD = 128;
// Info bytes per edge in a matched tree
private static final int EDGE_BYTES = 4;
// Maximum content dissimilarity when fuzzmatching nodes in dfs search
private final static double DFS_MATCH_THRESHOLD = 0.2;
// Maximum content dissimilarity when searching for potential matching
// subtree roots.
private static final double MAX_FUZZY_MATCH = 0.2;

private void removeSmallCopies( BranchNode root ) {
  BaseNode base = root.getBaseMatch();
  if( base != null && base.getLeft().getMatches().size() > 1 ) {
    // Iterate over the matches, and discard any that too small
    Set deletia = new HashSet();
    for( Iterator i = base.getLeft().getMatches().iterator();i.hasNext();) {
      BranchNode copy = (BranchNode) i.next();
      if( copy.getMatchArea().getInfoBytes() < COPY_THRESHOLD ) {
          deletia.add(copy);
      }
    }
    if( base.getLeft().getMatches().size() == deletia.size() ) {
      // We're deleting all matches... check if some match is the "original"
      // instance and if it's found, don't delete it!
      int maxcopybytes = 0, mincopybytes = Integer.MAX_VALUE;
```



```
      BranchNode origInstance = null;
      for( Iterator i = base.getLeft().getMatches().iterator();i.hasNext();) {
        BranchNode copy = (BranchNode) i.next();
        int copybytes = copy.getMatchArea().getInfoBytes();
        Node copyRoot = copy.getMatchArea().getRoot();
        Node copyBase = ((BranchNode) copyRoot).getBaseMatch();
        // Scan left
        while(  (copyRoot =  copyRoot.getLeftSibling()) != null &&
               (copyBase =  copyBase.getLeftSibling()) != null &&
               ((BranchNode) copyRoot).getBaseMatch() == copyBase &&
               copybytes < COPY_THRESHOLD)
          copybytes += ((BranchNode) copyRoot).getMatchArea().getInfoBytes();
        // Scan right
        copyRoot = copy.getMatchArea().getRoot();
        copyBase = ((BranchNode) copyRoot).getBaseMatch();
        while( (copyRoot = (BranchNode) copyRoot.getRightSibling()) != null &&
               (copyBase = (BaseNode) copyBase.getRightSibling()) != null &&
               ((BranchNode) copyRoot).getBaseMatch() == copyBase &&
               copybytes < COPY_THRESHOLD)
          copybytes += copyRoot.getMatchArea().getInfoBytes();
        if( copybytes > maxcopybytes ) {
          origInstance = copy;
          maxcopybytes = copybytes;
        }
        if( copybytes < mincopybytes )
          mincopybytes = copybytes;
      }
      if( maxcopybytes > mincopybytes ) {
        // Keep matching if there is a copy that is "better" (more copybytes)
        deletia.remove(origInstance);
        // Mark match as original instance by ensuring its subtree ha more
        // info than COPY_THRESHOLD (=> won't be unamtched anymore)
        origInstance.getMatchArea().addInfoBytes(COPY_THRESHOLD+1);
      }
    }
    if( !deletia.isEmpty() ) {
      for( Iterator i = deletia.iterator();i.hasNext();)
        delMatchArea(((BranchNode) i.next()).getMatchArea());
    }
  }
  for(int i=0;i<root.getChildCount();i++)
    removeSmallCopies(root.getChild(i));
}


/** Find matching candidates for a node. */
protected Vector findCandidates( BaseNode tree, BranchNode key ) {
  Vector candidates = new Vector();
  findExactMatches( tree, key, candidates );
  if( candidates.isEmpty() )
    findFuzzyMatches( tree, key, candidates );
  return candidates;
}

protected int dfsMatch( BaseNode a, BranchNode b, int count ) {
  return dfsMatch(a,b,count,null,null);
}

/** Match truncated subtrees. Matches two subtrees using dfs. If stopNodes is
 *  null, it only calculates how many nodes would be matched if the function
 *  was run "for real".
 *  @param a Root of the base truncated matching subtree
 *  @param b Root of the branch truncated matching subtree
 *  @param stopNodes Vector to store the children of the leaf nodes of the
 *                truncated matching subtree. Filled in by the procedure.
 *                If set to <b>null</b> no matches between nodes are added,
 *                only the node count is calculated.
```



```
 *  @param ma used to tag the matched nodes. The infobytes of <code>ma</co
de>
 *  are updated to the size of the matching subtree by the function.
 *  @return number of nodes in the matching subtree.
 */
protected int dfsMatch( BaseNode a, BranchNode b, int count, Vector stopNod
es,
                  MatchArea ma ) {
  if( stopNodes != null ) {
    // Also means matchings should be added
    addMatching( b, a );
    ma.addInfoBytes( a.getContent().getInfoSize() );
    b.setMatchArea( ma );
  }
  boolean childrenMatch = true;
  if( a.getChildCount() == b.getChildCount() ) {
    // Only match children, if there are equally many
    for( int i=0; childrenMatch && i<a.getChildCount(); i ++ ) {
      childrenMatch = a.getChild(i).getContent().contentEquals(
        b.getChild(i).getContent());
    }
  } else
    childrenMatch = false;

  if( !childrenMatch ) {
    // Mark all children as stopnodes
    for( int i=0; stopNodes!=null && i<b.getChildCount(); i ++ )
      stopNodes.add( b.getChild(i) );
  } else {
    // All children match
    if( ma != null )
      ma.addInfoBytes( a.getChildCount() * EDGE_BYTES );
    for( int i=0; i<a.getChildCount(); i ++ )
      count += dfsMatch( a.getChild(i), b.getChild(i), 0, stopNodes, ma );
  }
  return count + 1;
}

/** Get the children the leaf nodes of a matching subtree. Uses the
 *  subtree tag of the nodes (MatchArea class) */
public void getAreaStopNodes( Vector stopNodes, BranchNode n ) {
  boolean childrenInSameArea = true;
  MatchArea parentArea = n.getMatchArea();
  if( parentArea == null )
    throw new RuntimeException("ASSERT FAILED");
  for( int i=0;i<n.getChildCount() && childrenInSameArea;i++)
    childrenInSameArea&= n.getChild(i).getMatchArea() == parentArea;
  if( !childrenInSameArea ) {
    stopNodes.add(n);
    return;
  } else {
    for( int i=0;i<n.getChildCount();i++)
      getAreaStopNodes(stopNodes,n.getChild(i));
  }
}

protected boolean dfsTryFuzzyMatch( Node a, Node b) {
  double distance = measure.getDistance(a,b );
  return  distance < DFS_MATCH_THRESHOLD;
}

protected void findExactMatches( BaseNode tree, BranchNode key,
                         Vector candidates ) {
  for( Iterator i = new DFSTreeIterator(tree);i.hasNext();) {
    BaseNode cand = (BaseNode) i.next();
    if( cand.getContent().contentEquals(key.getContent()) )
      candidates.add( new CandidateEntry( cand, 0.0,
```

```
        -1.0 )); // -1.0 => lr value not present
  }
}


/** Find fuzzy matches for a node. The fuzzy matches found are candidate
 *  roots for matching subtrees.
 */
protected void findFuzzyMatches( BaseNode tree, BranchNode key,
                                 Vector candidates ) {
  SortedSet cset = new TreeSet(candComp);
  double cutoff = 2*MAX_FUZZY_MATCH;
  for( Iterator i = new DFSTreeIterator(tree);i.hasNext();) {
    BaseNode cand = (BaseNode) i.next();
    double discount = 1.0;
    double lrdDist = discount*Math.min(
      Math.min(getDistanceOfLeft(key,cand),getDistanceOfRight(key,cand)),
      measure.childListDistance(key,cand));
    double minDist = discount*Math.min(lrdDist,measure.getDistance(cand,key))
;
    if( minDist < 2*MAX_FUZZY_MATCH ) {
      cset.add(new CandidateEntry(cand,minDist,lrdDist));
      cutoff = cutoff < 2*minDist ? cutoff : 2*minDist;
    }
  }
  for( Iterator i = cset.iterator();i.hasNext();) {
    CandidateEntry en = (CandidateEntry) i.next();
    if( en.distance > cutoff )
      break;
    else
      candidates.add(en);
  }
}

// Distance value used by getDistanceOfleft/Right if both  nodes are at the
// end of the child list where no such node exists; e.g. getLeft and both
// nodes are at position 0.
private static final double END_MATCH = Measure.MAX_DIST;

private static Measure measure = new Measure();

/** Get content distance of left siblings. */
protected double getDistanceOfLeft( Node a, Node b ) {
  if( a.parent == null || b.parent == null )
    return Measure.MAX_DIST;
  if( a.getChildPos() > 0 && b.getChildPos() > 0 ) {
    return measure.getDistance(a.getLeftSibling(), b.getLeftSibling() );
  } else
    return END_MATCH;
}

/** Get content distance of right siblings. */
protected double getDistanceOfRight( Node a, Node b ) {
  if( a.parent == null || b.parent == null )
    return Measure.MAX_DIST;
  if( a.getChildPos() < a.parent.getChildCount() -1 &&
          b.getChildPos() < b.parent.getChildCount() -1 ) {
    return measure.getDistance(a.getRightSibling(), b.getRightSibling() );
  } else
    return END_MATCH;
}

/** Add Matching. Only adds the matching if types match (i.e. both text or
  element) */
protected void addMatchingIfSameType( BranchNode a, BaseNode b ) {
  if( (a.getContent() instanceof XMLTextNode &&
       b.getContent() instanceof XMLTextNode) ||
```



```
       (a.getContent() instanceof XMLElementNode &&
        b.getContent() instanceof XMLElementNode)) {
    addMatching(a,b);
  }
}

protected void addMatching( BranchNode a, BaseNode b ) {
  a.setBaseMatch(b,BranchNode.MATCH_FULL);
  b.getLeft().addMatch(a);
}

protected void delMatching( BranchNode a, BaseNode b ) {
  a.delBaseMatch();
  b.getLeft().delMatch(a);
}

protected void delMatchArea(MatchArea m) {
  delMatchArea(m.getRoot(),m);
}

/** Remove subtree tag from a matched subtree.
 *  @param n root of removal
 *  @param m subtree tag (=tag of root) */
private void delMatchArea(BranchNode n,MatchArea m) {
  if( n.getMatchArea() == m ) {
    n.setMatchArea(null);
    delMatching(n,n.getBaseMatch());
    for( int i=0;i<n.getChildCount();i++)
      delMatchArea(n.getChild(i),m);
  }
}

private boolean isMatched( BaseNode n) {
  return n.getLeft().getMatchCount() > 0;
}

/** Class for match candidates. Stores some distance measures in addition to
the node itself. */
class CandidateEntry {
  BaseNode candidate=null;
  double leftRightDown = 0.0;
  double distance=0.0;
  CandidateEntry( BaseNode n, double d, double lrd ) {
    candidate = n;
    distance = d;
    leftRightDown = lrd;
  }
}

// Used when sorting matches. Sorts by distance
private Comparator candComp = new Comparator() {
  public int compare(Object o1, Object o2) {
    double diff = ((CandidateEntry) o1).distance-((CandidateEntry) o2).distance;
    return diff < 0 ? -1 : (diff> 0 ? 1 : 0 );
  }
};

// Used when sorting matches. Sorts by leftRightDown distances
private Comparator candlrdComp = new Comparator() {
  public int compare(Object o1, Object o2) {
    double diff = ((CandidateEntry) o1).leftRightDown-((CandidateEntry) o2).left
RightDown;
    return diff < 0 ? -1 : (diff> 0 ? 1 : 0 );
  }
};

// Iterator for iterating over a tree in DFS order.
```



```
  class DFSTreeIterator implements Iterator {
    Node currentNode = null;
    int currentChild = 0;

    public DFSTreeIterator( Node root ) {
      currentNode = root;
    }

    public boolean hasNext() {
      return currentNode != null;
    }

    public Object next() {
      Node result = currentNode;
      if( currentNode.getChildCount() > 0 )
        currentNode = currentNode.getChildAsNode(0);
      else  {
       // back up until unvisited child found
        while( currentNode != null &&
          ( currentNode.parent == null || currentNode.childPos == currentNode.pare
nt.getChildCount()-1 ) )
          currentNode = currentNode.parent;
        if ( currentNode != null )
          currentNode = currentNode.parent.getChildAsNode( currentNode.childPos
+ 1 );
        }
      return result;
    }

    public void remove()  {
      throw new java.lang.UnsupportedOperationException();
    }
  }
}
```

```java
//$Id: Measure.java,v 1.8 2001/09/05 21:22:28 ctl Exp $ D

import org.xml.sax.Attributes;
import java.util.Map;
import java.util.HashMap;
import java.util.Iterator;

/** Node similarity measurement. This class is used to calculate the
 *  content, child list and matched child list distance between nodes.
 */

public class Measure {

  // Maximum distance. The distance is normalized between 0 and MAX_DIST
  public static final double MAX_DIST = 1.0;
  // Distance to return by childListDistance if both nodes have 0 children
  public static final double ZERO_CHILDREN_MATCH = 1.0;
  // info bytes in an element name ($c_e$ in thesis)
  public static final int ELEMENT_NAME_INFO = 1;
  // info bytes in the presence of an attribute ($c_a$ in thesis)
  public static final int ATTR_INFO = 2;
  // attribute values less than this have a info size of 1 ($c_v$ in thesis)
  public static final int ATTR_VALUE_THRESHOLD = 5;
  // text nodes shorter than this have an info size of 1 ($c_t$ in thesis)
  public static final int TEXT_THRESHOLD = 5;

  public Measure() {
  }

  // State variables. State is used internally to allow the class to be extended
  // to calculate the distance between the combined contents of several nodes.
  // Currently, only parwise distance calculation is visible outside the class.

  // Mismatched info bytes
  private int mismatched = 0;
  // Total info bytes
  private int total = 0;
  // set to true if total mismatch occurs (e.g. text and element node compared)
  private boolean totalMismatch = false;
  // penalty term ($c_p in thesis)
  private static final int C= 20;

  /** Return content distance between nodes. */
  public double getDistance(Node a, Node b ) {
    if( a!=null && b!= null )
      includeNodes( a, b );
    double penalty = Math.max(0.0,1.0-((double) total)/((double) C));
    double distance= penalty+(1.0-penalty)*((double) (mismatched))/
                   ((double) total);
    resetDistance();
    return totalMismatch ? 1.0 : distance;
  }

  // Currently not used. used to read distance without adding more nodes
  private double getDistance() {
    return getDistance(null,null);
  }

  private void resetDistance() {
    mismatched = 0;
    total = 0;
    totalMismatch = false;
  }

  // Add node pair internal to state
  private void includeNodes( Node a, Node b ) {
    if( a== null || b== null || totalMismatch)
```



```java
      return;
    XMLNode ca = a.getContent(), cb = b.getContent();
    if( ca instanceof XMLElementNode && cb instanceof XMLElementNode ) {
      XMLElementNode ea = (XMLElementNode) ca,eb=(XMLElementNode) cb;
      total+=ELEMENT_NAME_INFO;
      mismatched += ea.getQName().equals(eb.getQName()) ? 0 : ELEMENT_NA
ME_INFO;
      Attributes aa =  ea.getAttributes(), ab = eb.getAttributes();
      for( int i=0;i<aa.getLength();i++)  {
        int index = ab.getIndex(aa.getQName(i));
        if( index != -1 ) {
          String v1 =  aa.getValue(i), v2 = ab.getValue(index);
          int amismatch = stringDist( v1,v2,1.0 );
          int info = (v1.length() > ATTR_VALUE_THRESHOLD ? v1.length() : 1 ) +
                  (v2.length() > ATTR_VALUE_THRESHOLD ? v2.length() : 1 );
          mismatched += amismatch > info ? info : amismatch;
          total+=info;
        } else {
          mismatched += ATTR_INFO;
          total += ATTR_INFO;
        }
      }
      // Scan for deleted from b
      for( int i=0;i<ab.getLength();i++) {
        if( aa.getIndex(ab.getQName(i)) == -1 ) {
          mismatched += ATTR_INFO;
          total += ATTR_INFO;
        }
      }
    } else if ( ca instanceof XMLTextNode && cb instanceof XMLTextNode ) {
      int info = ca.getInfoSize() + cb.getInfoSize() / 2,
        amismatch = stringDist( ((XMLTextNode) ca).getText(),
                       ((XMLTextNode) cb).getText(),1.0 ) / 2;
      mismatched += amismatch > info  ? info : amismatch;
      total+=info;
    } else
      totalMismatch = true;
  }

  public int stringDist( String a, String b,double limit ) {
    return qDist(a,b);
  }

  public int stringDist( char[] a, char[] b,double limit ) {
    return qDist(a,b);
  }

  public double childListDistance( Node a, Node b ) {
    if( a.getChildCount()== 0 && b.getChildCount() == 0)
      return ZERO_CHILDREN_MATCH; // Zero children is also a match!
    else {
      char[] ac = new char[a.getChildCount()];
      char[] bc = new char[b.getChildCount()];
      for( int i=0;i<a.getChildCount();i++)
        ac[i]=(char) (a.getChildAsNode(i).getContent().getContentHash()&0xffff);
      for( int i=0;i<b.getChildCount();i++)
        bc[i]=(char) (b.getChildAsNode(i).getContent().getContentHash()&0xffff);
      return ((double) stringDist(ac,bc,1.0))
                / ((double) a.getChildCount() + b.getChildCount());
    }
  }

  public int matchedChildListDistance( BaseNode a, BranchNode b ) {
    char[] ac = new char[a.getChildCount()];
    char[] bc = new char[b.getChildCount()];
    // NOTE! i+1's as 0=-0! (Which would yield equality for all child lists of
    // length 1). Using i instead of i+1 was a bug that took some time to find
```



```java
    for( int i=0;i<a.getChildCount();i++)
      ac[i]=(char) (i+1);
    for( int i=0;i<b.getChildCount();i++) {
      BaseNode m = b.getBaseMatch();
      if( m!= null && m.getParent() == a )
        bc[i] = (char) m.getChildPos();
      else
        bc[i] = (char) -(i+1);
    }
    return stringDist(ac,bc,1.0);
  }


  // q-Gram Distance [Ukkonen92]
  // The implementation could use some heavy optimization. It's really a
  // textbook example of inefficient coding...

  class Counter {
    public int count = 1;
  }

  private static final int INIT_CAPACITY=2048;

  // Which gram to use. Yes, I know its ugly to have as
  // a state var, as it changes. But it origibally didn't :)
  private static int Q=4;

  // Hsh tables used to store q-Grams
  private Map aGrams = new HashMap(INIT_CAPACITY);
  private Map bGrams = new HashMap(INIT_CAPACITY);

  protected int qDist( String a , String b ) {
    decideQ(a.length()+b.length());
    buildQGrams(a,aGrams);
    buildQGrams(b,bGrams);
    return calcQDistance();
  }

  protected int qDist( char[] a , char[] b ) {
    decideQ(a.length+b.length);
    buildQGrams(a,aGrams);
    buildQGrams(b,bGrams);
    return calcQDistance();
  }

  protected void buildQGrams(String a, Map grams) {
    grams.clear();
    for( int i=0;i<a.length();i++) {
    String gram = a.substring(i,i+Q > a.length() ? a.length() : i+Q );
    if( grams.containsKey(gram) )
      ((Counter) grams.get(gram)).count++;
    else
      grams.put(gram,new Counter());
    }
  }

  protected void buildQGrams(char[] a, Map grams) {
    grams.clear();
    for( int i=0;i<a.length;i++) {
      int count = i + Q > a.length ? a.length -i : Q;
      String gram = new String(a,i,count);
      if( grams.containsKey(gram) )
        ((Counter) grams.get(gram)).count++;
      else
        grams.put(gram,new Counter());
    }
  }
```

Measure.java **Page 4 of 4**

```
  // Decide with q-gram to use, based on the length of the content
  protected int decideQ( int total ) {
   int q = 1;
   if( total > 150 )
     q = 4;
    else if( total > 50 )
     q = 2;
    return q;
  }

  protected int calcQDistance() {
    int dist = 0;
    // first, loop over agrams
    for( Iterator i = aGrams.keySet().iterator();i.hasNext();) {
      Object gramA = i.next();
      int countA = ((Counter) aGrams.get(gramA)).count;
      int countB = 0;
      if( bGrams.containsKey(gramA) )
        countB = ((Counter) bGrams.get(gramA)).count;
      else
      dist += Math.abs(countA-countB);
    }
    // And add any grams present in b but not in a
    for( Iterator i = bGrams.keySet().iterator();i.hasNext();) {
      Object gramB = i.next();
      if( !aGrams.containsKey(gramB) ) {
        dist += ((Counter) bGrams.get(gramB)).count;
      }
    }
    return dist;
  }
}
```

Merge.java **Page 1 of 15**

```
// $Id: Merge.java,v 1.36 2001/09/05 21:22:29 ctl Exp $ D

import org.xml.sax.ContentHandler;
import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.AttributesImpl;
import java.util.Vector;
import java.util.Set;
import java.util.HashSet;
import java.util.Map;
import java.util.HashMap;
import java.util.Iterator;
import java.util.NoSuchElementException;

/** 3-way merge of a base and two branch trees. */

public class Merge {

  private TriMatching m = null;
  private PathTracker pt = new PathTracker();
  private ConflictLog clog = new ConflictLog(pt);
  private EditLog elog = new EditLog(pt);

  /** Construct a new merge object. The matched trees to merge are passed in
   *  the TriMatching argument.
   */

  public Merge(TriMatching am) {
    m = am;
  }

  public ConflictLog getConflictLog() {
    return clog;
  }

  public EditLog getEditLog() {
    return elog;
  }

  /** Run 3-way merge of trees */
  public void merge( ContentHandler ch ) throws SAXException {
    pt.resetContext();
    ch.startDocument();
    treeMerge( m.getLeftRoot(), m.getRightRoot(), ch );
    ch.endDocument();
  }

  /** Main merging function. Merges the child lists of a and b, and then
   *  recurses for each child in the merged list.
   */

  protected void treeMerge( BranchNode a, BranchNode b, ContentHandler ch )
                          throws SAXException {
    MergeList mlistA = a != null ? makeMergeList( a ) : null;
    MergeList mlistB = b != null ? makeMergeList( b ) : null;
    MergePairList merged = null;
    pt.enterSubtree(); // Update path tracker to tell where we are...
    // Generate merge pair List, store in the merged object
    if( mlistA != null && mlistB != null )
      merged = makeMergePairList( mlistA, mlistB ); // Merge lists
    else
      // Only one child list, construct mergepairlist directly from child mlist
      merged = mergeListToPairList( mlistA == null ? mlistB : mlistA, null );
    // Handle updates & Recurse
    for( int i=0;i<merged.getPairCount();i++) {
      MergePair mergePair = merged.getPair(i);
      XMLNode mergedNode = mergeNodeContent( mergePair );
```

Merge.java **Page 10 of 15**

```
  /** Remove deleted or far moved nodes from merge lists. */
  private void removeDeletedOrMoved( MergeList mlistA, MergeList mlistB ) {
    BaseNode baseParent = mlistA.getEntryParent().getBaseMatch();
    for( int i=0;i<baseParent.getChildCount();i++) {
      BaseNode bn = baseParent.getChild(i);
      int op1 = getOperation( bn, mlistA ),
          op2 = getOperation( bn, mlistB );
      // Swap ops, so that op1 is always the smaller (to simplify the if clauses)
      if( op1 > op2 ) {
        int t=op1; op1=op2; op2=t;
        MergeList tl = mlistA; mlistA = mlistB; mlistB = tl;
      }
      /*************************************************************
      * Table to implement; mlistA is for op1 and mlistB for op2
      * Op1     Op2
      * NOP     NOP     OK
      * NOP     MOVE_I  OK, internal moves are handled by merging of the lists
      * NOP     MOVE_F  Delete the node from mlistA
      * NOP     DELETE  Delete the node from mlistA
      * MOVE_I  MOVE_I  OK, internal moves are handled by merging of the lists
      * MOVE_I  MOVE_F  Conflicting moves
      * MOVE_I  DELETE  Conflict - node is deleted and moved
      * MOVE_F  MOVE_F  Possibly conflict, see the code below
      * MOVE_F  DELETE  Conflict - node is deleted and moved
      * DELETE  DELETE  OK
      **************************************************************/
      if( (op1==NOP && op2==NOP ) ||
          (op1==NOP && op2==MOVE_I ) ||
          (op1==MOVE_I && op2==MOVE_I ) ||
          (op1==DELETE && op2==DELETE ) )
        continue; // All OK cases
      if( op1 == NOP && ( op2 == MOVE_F || op2 == DELETE ) ) {
        // Delete the node from mlistA
        int ix = mlistA.matchInList(bn);
        if( op2==DELETE && isDeletiaModified(mlistA.getEntry(ix).getNode(),
            mlistA) )
          // CONFLICTCODE
          clog.addListWarning( ConflictLog.DELETE, "Modifications in "+
          "deleted subtree.",bn,mlistA.getEntry(ix).getNode(),null);
        if( mlistA.getEntry(ix).getHangonCount() > 0 ) {
          // we need to move the hangons to the predecessor
          for( int ih = 0; ih < mlistA.getEntry(ix).getHangonCount(); ih++)
            mlistA.getEntry(ix-1).addHangon(mlistA.getEntry(ix).getHangon(ih));
        }
        int matchIx = mlistA.matchInList(bn);
        if( op2==DELETE )
          elog.delete(mlistA.getEntry(matchIx).getNode().getBaseMatch(),mlistB.get
EntryParent());
        if( op2==DELETE && mlistA.getEntry(matchIx).locked ) {
          clog.addListConflict(ConflictLog.DELETE, "Moved or copied node " +
          "deleted. Moving on by allowing the delete.", bn,
          mlistA.getEntry(ix).getNode(),null);
        }
        mlistA.removeEntryAt(matchIx);
      } else if( op1 == MOVE_I && op2== MOVE_F ) {
        // CONFLICTCODE
        BranchNode op1node = mlistA.getEntry( mlistA.matchInList(bn)).getNode();
        clog.addListConflict( ConflictLog.MOVE, "Node moved to different "+
        "locations - trying to recover by ignoring move inside childlist "+
        "(copies and inserts immediately following the node may have been "+
        "deleted)",bn,op1node,op1node.getFirstPartner(BranchNode.MATCH_FUL
L));
        mlistA.removeEntryAt(mlistA.matchInList(bn));
      } else if( op1 == MOVE_I && op2== DELETE ) {
        // CONFLICTCODE
        clog.addListConflict(ConflictLog.MOVE,"Node moved and deleted - trying
"+
```

```
        " to recover by deleting the node (copies and inserts immediately "+
        "following the node may also have been deleted)",bn,
        mlistA.getEntry( mlistA.matchInList(bn) ).getNode(), null );
        int matchIx = mlistA.matchInList(bn);
        elog.delete(mlistA.getEntry(matchIx).getNode().getBaseMatch(),
        mlistB.getEntryParent());
        mlistA.removeEntryAt(matchIx);
      } else if( op1 == MOVE_F && op2 == MOVE_F ) {
        if( isMovefMovefConflict( bn ) ) {
          // CONFLICTCODE
          clog.addListConflict( ConflictLog.MOVE,"The node was moved to "+
          "different locations. It will appear at each location.", bn,
          bn.getLeft().getFullMatch(),bn.getRight().getFullMatch() );
        }
      } else if (op1 == MOVE_F && op2 == DELETE ) {
          // CONFLICTCODE here
          clog.addListConflict( ConflictLog.MOVE,"The node was moved and "+
          "deleted. Ignoring the deletion.", bn,bn.getLeft().getFullMatch(),
          bn.getRight().getFullMatch() );
      }
    }
  }

  // Check if the deletia rooted at n is modified w.r.t. base

  private boolean isDeletiaModified(BranchNode n, MergeList ml) {
    BaseNode m = n.getBaseMatch();
    if( m == null )
      return true; // the node was inserted => modified
    if( getOperation(m,ml) != NOP )
      // Notice that we check for move instantly, but updates only when
      // we know the node was deleted. This is because moves are
      // visible on the previous tree level compared to the updates
      return true; // the node has been moved
    if( n.getBaseMatchType() != BranchNode.MATCH_FULL )
      return true;
    // either structural or content modification (otherwise match would be full)
    // NOTE: By definition of a natural matching, at least one match is full!
    boolean deletedInOther = n.getPartners().getMatches().isEmpty();
    if( deletedInOther ) {
      if( !matches( n, m ) )
        return true; // The node is updated
      // Check children
      MergeList mlistN = makeMergeList(n);
      for( int i=0;i<n.getChildCount();i++) {
        if( isDeletiaModified( n.getChild(i), mlistN ) )
          return true;
      }
      return false; // got trough the children (recursively), no modifications
    } else
      // No modification here, and no recurse needed (the node has a structmatch
      // in the other tree)
      return false;
  }

  // Check a base node for a movef-movef conflict
  private boolean isMovefMovefConflict( BaseNode n ) {
    return _isMovefMovefConflict( n, n.getRight().getMatches(),
          n.getLeft().getMatches() ) || _isMovefMovefConflict( n,
          n.getLeft().getMatches(), n.getRight().getMatches() );
  }

  // Check if base node n is moved under matching parents. If so, that is good,
  // because the list merge will handle possible conflicts. If they are not
  // moved under matching parents we have a conflict. (which unresolved results
  // in two copies). Specifically the condition is that the parent of each
  // BranchNode bn is moved to (structurally or content) must structurally match
```



```
  // another BranchNode that has bn as a child.

  // Although called rarely, this code is really horribly slow --- O(n^3)?!
  // Some easy optimizations hsould be possible. Also note that it is not
  // included in the complexity analysis of the algorithm

  // Final note: the author suspects this code might have a bug or few!
  private boolean _isMovefMovefConflict( BaseNode n, Set matchesA,
                                        Set matchesB ) {
    for( Iterator i = matchesB.iterator(); i.hasNext(); ) {
      BranchNode bnA = (BranchNode) i.next();
      BranchNode bnAparent = bnA.getParent();
      if( (bnAparent.getBaseMatchType() & BranchNode.MATCH_CHILDREN) ==
0)
        // here's a copy with no structural match on the other side => conflict
        return true;
      for( Iterator ip = bnAparent.getPartners().getMatches().iterator();
                                        ip.hasNext(); ) {
        BranchNode bnBparent = (BranchNode) ip.next();
        boolean hasNasChild = false;
        for( int ic = 0; ic < bnBparent.getChildCount() && !hasNasChild;ic ++ )
          hasNasChild = matchesA.contains( bnBparent.getChild(ic) );
        if( hasNasChild && ( bnBparent.getBaseMatchType() &
                  BranchNode.MATCH_CHILDREN ) == 0 )
          return true; // here's a copy with no structural match on the other side => c
onflict
      }
    }
    return false;
  }

  // Container for hangons
  private class HangonEntry {
    BranchNode node = null;
    HangonEntry( BranchNode an ) {
      node = an;
    }

    HangonEntry() {
    }

    public BranchNode getNode() {
      return node;
    }

    public String toString() {
      return node.getContent().toString();
    }
  }

  class MergeEntry extends HangonEntry {

    Vector inserts = new Vector();
    boolean locked = false;
    private BranchNode mergePartner = null;
    private boolean moved = false;

    MergeEntry( BranchNode n) {
      super(n );
    }

    MergeEntry() {
    }

    public boolean isMoved() {
      return moved;
    }
```



```
    public void setMoved(boolean amoved ) {
      moved=amoved;
    }

    public void setMergePartner(BranchNode n) {
      mergePartner = n;
    }

    public BranchNode getMergePartner() {
      return mergePartner;
    }

    int getHangonCount() {
      return inserts.size();
    }

    HangonEntry getHangon( int ix ) {
      return (HangonEntry) inserts.elementAt(ix);
    }

    void addHangon( BranchNode n  ) {
      addHangon( new HangonEntry( n ) );
    }

    void addHangon( HangonEntry e ) {
      inserts.add( e );
    }

  }

  // Merge list start and end markers. Cleaner if they were in MergeList, but
  // Java doesn't allow statics in nested classes
  static BranchNode START = new BranchNode(new XMLTextNode("__START
__"));
  static BranchNode END = new BranchNode(new XMLTextNode("__END__"));

  // TODO: START end END markers should be completely hidden if possible
  class MergeList {
    private Vector list = new Vector();
    // lokkup table: look up Entry index based on base partner
    private Map index = new HashMap();
    private int tailPos = -1; // current tail pos
    private BranchNode entryParent = null; // Common parent of all entries
    private MergeEntry currentEntry = null;

    public MergeList( BranchNode anEntryParent ) {
      entryParent = anEntryParent;
    }

    public BranchNode getEntryParent() {
      return entryParent;
    }

    public void add( MergeEntry n ) {
      tailPos++;
      ensureCapacity( tailPos + 1, false);
      if( list.elementAt(tailPos) != null )
        n.locked = ((MergeEntry) list.elementAt(tailPos)).locked;
      list.setElementAt(n,tailPos);
      index.put( n.node.getBaseMatch(), new Integer(tailPos));
      currentEntry = n;
    }

    void add( BranchNode n ) {
      add( new MergeEntry(n ) );
    }
```

```
  void addHangOn( BranchNode n ) {
    getEntry(tailPos).addHangon(n );
    currentEntry = null;
  }

  public void setMoved( boolean moved ) {
    currentEntry.setMoved( moved );
  }

  public int getEntryCount() {
    return tailPos + 1;
  }

  public MergeEntry getEntry( int ix ) {
    return (MergeEntry) list.elementAt(ix);
  }

  // OPTIMIZATION FIX: The method needs to rebuild the index-- this should be
  // be optimized away with e.g. lazy deletions and an aggregated index
  // rebuild. Or maybe we could make the index use pointers.
  public void removeEntryAt( int ix ) {
    list.removeElementAt(ix);
    tailPos--;
    index.clear();
    for( int i=0;i<getEntryCount();i++)
      index.put( getEntry(i).node.getBaseMatch(), new Integer(i));
  }

  public void lockNeighborhood(  int left, int right ) {
    lockNeighborhood( tailPos, left, right );
  }

  public void lockNeighborhood( int acurrentPos, int left, int right ) {
    ensureCapacity( acurrentPos + right + 1, true);
    for( int i = acurrentPos - left; i<=acurrentPos + right; i++)
      ((MergeEntry) list.elementAt(i)).locked = true;
  }

  public int findPartner( MergeEntry b ) {
    if( b.node == START )
      return 0;
    else if( b.node == END )
      return getEntryCount() - 1; // Assuming the other list is equally long
    return ((Integer) index.get( b.node.getBaseMatch() )).intValue();
  }

  public int matchInList( BaseNode n ) {
    Integer i = (Integer) index.get( n );
    if( i == null )
      return -1;
    else
      return i.intValue();
  }

  private void ensureCapacity( int size, boolean fill ) {
    for( int i = list.size(); i < size; i ++ )
      list.add( fill ? new MergeEntry() : null);
  }
}

//
//
// Utility functions
//
protected boolean matches( Node a, Node b ) {
  if( a== null || b==null)
```



```
    return false;
  return a.getContent().contentEquals(b.getContent());
}

/** Check if entire subtrees match exactly. */
protected boolean treeMatches( Node a, Node b ) {
  if( !matches(a,b) )
    return false;
  if( a.getChildCount() != b.getChildCount() )
    return false;
  boolean matches = true;
  for( int i=0;i<a.getChildCount() && matches;i++)
    matches &= treeMatches(a.getChildAsNode(i),b.getChildAsNode(i));
  return matches;
}
}
```



```
      if( mergedNode instanceof XMLTextNode ) {
        XMLTextNode text = (XMLTextNode) mergedNode;
        ch.characters(text.getText(),0,text.getText().length);
        // NOTE: Theoretically, if we have matched text and element nodes we
        // need to recurse here. But, the current matching algo never matches
        // across types, so there's no need for recursion
      } else {
        // It's an element node
        XMLElementNode mergedElement = (XMLElementNode) mergedNode;
        ch.startElement(mergedElement.getNamespaceURI(),
                mergedElement.getLocalName(),mergedElement.getQName(),
                mergedElement.getAttributes());
        // Figure out partners for recurse
        MergePair recursionPartners = getRecursionPartners( mergePair );
        // Recurse for subtrees
        treeMerge(recursionPartners.getFirstNode(),
             recursionPartners.getSecondNode(),ch);
        ch.endElement(mergedElement.getNamespaceURI(),
               mergedElement.getLocalName(),
               mergedElement.getQName());
      }
      pt.nextChild();
    }
    pt.exitSubtree();
  }

  /** Return merged content of the nodes in a merge pair. */
  protected XMLNode mergeNodeContent( MergePair mp ) {
    // Merge contents of node and partner (but only if there's a partner)
    // -------------------
    // Table
    // n1    n2     Merge
    // any   null   n1
    // cont  cont   merge(n1,n2)
    // cont  str    n2
    // cont  full   merge(n1,n2)
    // str   str    FORCED content merge
    // str   full   n1 cont
    // full  full   merge(n1,n2)

    BranchNode n1 = mp.getFirstNode(), n2 = mp.getSecondNode();
    if( n1 == null || n2==null )
      return (n1==null ? n2 : n1).getContent();
    else if( n1.isMatch(BranchNode.MATCH_CONTENT) ) {
      if( !n2.isMatch(BranchNode.MATCH_CONTENT) ) {
        logUpdateOperation(n2);
        return n2.getContent();
      } else
        return cmerge( n1, n2 );
    } else {
      // n1 doesn't match content
      if( n2.isMatch(BranchNode.MATCH_CONTENT) ) {
        logUpdateOperation(n1);
        return n1.getContent();
      } else // Neither matches content => forced merge
        return cmerge( n1, n2 );
    }
  }

  /** Get partners for recursion in the merge dchild list */
  protected MergePair getRecursionPartners(MergePair mp) {
    BranchNode n1 = mp.getFirstNode(), n2 = mp.getSecondNode();
    if( n1 == null || n2 == null ) {
      // No pair, so just go on!
      return mp;
    } else {
      // We have a pair, do as the table in the thesis says:
```

```
      // n1    n2     Merge
      // ------------------
      // any   -      n1,-
      // -     any    -,n2
      // str   str    n1,n2
      // str   cont   -,n2
      // cont  str    n1,-
      // cont  cont   n1,n2 (FORCED)
      if( n1.isMatch(BranchNode.MATCH_CHILDREN) &&
          n2.isMatch(BranchNode.MATCH_CHILDREN) )
        return mp;
      else if( n1.isMatch(BranchNode.MATCH_CHILDREN) &&
             n2.isMatch(BranchNode.MATCH_CONTENT) )
        return new MergePair(n2,null);
      else if( n1.isMatch(BranchNode.MATCH_CONTENT) &&
             n2.isMatch(BranchNode.MATCH_CHILDREN) )
        return new MergePair(n1,null);
      else // Both content matches --> forced merge
        return mp;
    }
  }

  /** Merge content of two nodes. */
  private XMLNode cmerge( BranchNode a, BranchNode b ) {
    boolean aUpdated = !matches( a, a.getBaseMatch() ),
            bUpdated = !matches( b, b.getBaseMatch() );
    if( aUpdated && bUpdated ) {
//        System.out.println(a.isLeftTree() + ": " + a.getContent().toString() );
//        System.out.println(b.isLeftTree() + ": " + b.getContent().toString() );
      if( matches( a, b ) ) {
        clog.addNodeWarning(ConflictLog.UPDATE,
          "Node updated in both branches, but updates are equal",
          a.getBaseMatch(),a,b);
        logUpdateOperation(a);
        return a.getContent();
      } else {
        XMLNode merged = null;
        // if XMLElementNode try merging attributes; if XMLTextnode give up
        if( a.getContent() instanceof XMLElementNode &&
            b.getContent() instanceof XMLElementNode ) {
          merged = mergeElementNodes(
            (XMLElementNode) a.getBaseMatch().getContent(),
            (XMLElementNode) a.getContent(), (XMLElementNode) b.getContent() );
        }
        if( merged != null )
          return merged;
        // XMLTextNodes, or failed to merge element nodes => conflict
        // CONFLICTCODE here
        clog.addNodeConflict(ConflictLog.UPDATE,
          "Node updated in both branches, using branch 1",a.getBaseMatch(),a,
b);
        if( a.isLeftTree() ) {
          logUpdateOperation(a);
          return a.getContent();
        } else {
          logUpdateOperation(b);
          return b.getContent();
        }
      }
    } else if ( bUpdated ) {
      logUpdateOperation(b);
      return b.getContent();
    } else if (aUpdated) {
      logUpdateOperation(a);
      return a.getContent();
    } else
      return a.getContent(); // none modified, a or b is ok
```



```
}

/** Merge content of two element nodes. */
private XMLElementNode mergeElementNodes( XMLElementNode baseN,
  XMLElementNode aN, XMLElementNode bN ) {
  String tagName ="";
  if( baseN.getQName().equals(bN.getQName()))
    tagName = aN.getQName();
  else if( baseN.getQName().equals(aN.getQName()))
    tagName = bN.getQName();
  else
    return null;//CONFLICT: Both changed (possibly same, but let's be careful)
  Attributes base = baseN.getAttributes(), a = aN.getAttributes(),
             b=bN.getAttributes();
  // Check deleted attribs
  Set deletia = new HashSet();
  for( int i=0;i<base.getLength();i++) {
    int ixa = a.getIndex(base.getQName(i));
    int ixb = b.getIndex(base.getQName(i));
    if((ixa == -1 && ixb!= -1 && !base.getValue(i).equals(b.getValue(ixb))) ||
       (ixb == -1 && ixa!= -1 && !base.getValue(i).equals(a.getValue(ixa))) )
      return null; // CONFLICTCODE: attrib deleted & changed
    if( ixa == -1 || ixb == -1 )
      deletia.add(base.getQName(i));
  }
  AttributesImpl merged = new AttributesImpl();
  // Build combined list (inserts from A + updated common in A & B)
  for( int i=0;i<a.getLength();i++) {
    String qName = a.getQName(i);
    String value = a.getValue(i);
    if( deletia.contains(qName) )
      continue; // was deleted
    int ixb = b.getIndex(qName);
    if( ixb == -1 )
      merged.addAttribute("","",qName,a.getType(i),value); // Insert
    else {
      String valueB = b.getValue(ixb);
      String valueBase = base.getValue(qName);
      if( valueB.equals(valueBase) )
        // A possibly updated
        merged.addAttribute("","",qName,a.getType(i),value);
      else if( value.equals(valueBase) )
        // B possibly updated
        merged.addAttribute("","",qName,b.getType(ixb),valueB);
      else
        // CONFLICT: Both changed (possibly same, but let's be careful)
        return null;
    }
  }
  // Insertions from b
  for( int i=0;i<b.getLength();i++) {
    String qName = b.getQName(i);
    if( deletia.contains(qName) || a.getIndex(qName) != -1)
      continue; // was deleted or already processed
    merged.addAttribute("","",qName,b.getType(i),b.getValue(i)); // Insert
  }
  return new XMLElementNode( tagName, merged );
}

/** Make merge list from the children of a node. */
protected MergeList makeMergeList( BranchNode parent ) {
  MergeList ml = new MergeList(parent);
  if( parent.getBaseMatch() == null ) {
    // The parent is unmatched, treat all nodes as inserts/n:th copies
    ml.add( START );
    for( int i = 0;i<parent.getChildCount();i++)
      ml.addHangOn( parent.getChild(i) );
```



```
    ml.lockNeighborhood(0,1);
    ml.add( END );
    return ml;
  }
  Map baseMatches = new HashMap();
  // Next is always prevChildPos + 1, so the first should be 0 =>
  // init to -1, -2 means not found in base
  int prevChildPos = -1;
  int childPos = -1;
  ml.add( START );
  for( int i = 0;i<parent.getChildCount();i++) {
    BranchNode current = parent.getChild(i);
    BaseNode match = current.getBaseMatch();
    if( match == null ) {
      // It's an insert node
      ml.addHangOn( current );
      ml.lockNeighborhood(0,1);
    } else if( match.getParent() != parent.getBaseMatch() ) {
      // Copied from elsewhere
      ml.addHangOn( current );
      ml.lockNeighborhood(0,1);
    } else if ( baseMatches.containsKey( match ) ) {
      // current is the n:th copy of a node (n>1)
      ml.addHangOn( current );
      Integer firstPos = (Integer) baseMatches.get(match);
      if( firstPos != null ) {
        // Lock the first occurenece as well
        ml.lockNeighborhood(firstPos.intValue(),1,1);
        // Put null into hashtable, so we won't lock more than once
        // (it wouldn't hurt, but just to be nice)
        baseMatches.put(match,null);
      }
      ml.lockNeighborhood(0,1);
    } else {
      // Found in base, check for moves
      ml.add( current );
      baseMatches.put( match, new Integer( ml.tailPos ) );
      childPos = match.getChildPos();
      childPos = childPos == -1 ? -2 : childPos; // Remember; not found = -2
      if( (prevChildPos + 1) != childPos ) {
        // Out of sequence, lock previous and this
        // e.g. -1 0 1 3 4 5 => 1 & 3 locked
        boolean moved = false;
        // Possibly out of sequence.. check of nodes between prev
        if( prevChildPos != -2 && childPos != -2 && prevChildPos < childPos ){
          // Not moved if every node between prevChildPos+1 and childPos-1
          // (ends included) is deleted
          // This code uses a simple for loop, instead of the fancy
          // in-sequence table described in the thesis. So far, tast enough in
          // the real world (tm)
          for(int j=0;!moved && j<parent.getChildCount();j++) {
            BaseNode aBase = parent.getChild(j).getBaseMatch();
            int basePos = aBase == null ? -1 : aBase.getChildPos();
            if( basePos != -1 && basePos > prevChildPos && basePos < childPos)
              moved = true;
          }
        } else
          moved = true;
        if( moved ) {
          ml.lockNeighborhood(1,0);
          ml.setMoved(true);
        } else
          ml.setMoved(false);
      }
      prevChildPos = childPos;
    } // end if found in base
  }
```

```
    ml.add( END );
    if( (prevChildPos + 1 )!= parent.getBaseMatch().getChildCount() )
      // Possible end out-of-seq; e.g. -1 0 1 2 4=e,
      // and 4 children in parent i.e. ix 3 was deleted
      ml.lockNeighborhood(1,0);
    return ml;
  }

  // Holds merge pairs
  class MergePair {
    BranchNode first,second;
    MergePair( BranchNode aFirst, BranchNode aSecond ) {
      first = aFirst;
      second = aSecond;
    }

    public BranchNode getFirstNode() {
      return first;
    }

    public BranchNode getSecondNode() {
      return second;
    }

  }

  // Merge pair list
  class MergePairList {
    Vector list = new Vector();
    public void append( BranchNode a, BranchNode b ) {
      list.add(new MergePair(a,b));
    }

    public int getPairCount() {
      return list.size();
    }

    public MergePair getPair(int ix){
      return (MergePair) list.elementAt(ix);
    }
  }

  /** Convert merge list to merge pair list. Also used as a fallback if
   *  node reordering fails in
   *  {@link #makeMergePairList(MergeList, MergeList)  mergePairList}.
   *  In that case, the hangons from <code>mlistB</code> are also
   *  used (i.e. the order is <code>mlistA</code> but any inserts
   *  copies and far moves in <code>mlistB</code> are included.
   *  @param mlistA merge list to convert
   *  @param mlistB merge list to take extra hangons from, null if none */
  protected MergePairList mergeListToPairList( MergeList mlistA,
                                     MergeList mlistB ) {
    // NOTE: We want to log all children of a non-structurally matched node as
    // copy/updates, That's why logHangonStructOps() is used throughout the func
    MergePairList merged = new MergePairList();
    for( int i=0;i<mlistA.getEntryCount()-1;i++) { // -1 due to end symbol
      MergeEntry me = mlistA.getEntry(i);
      if( i > 0) { // Don't append __START__
        merged.append(me.getNode(),
          me.getNode().getFirstPartner(BranchNode.MATCH_FULL));
        logHangonStructOps(me.getNode(),merged.getPairCount()-1);
      }
      for( int ih=0;ih<me.getHangonCount();ih++) {
        BranchNode hangon=me.getHangon(ih).getNode();
        merged.append(hangon,hangon.getFirstPartner(BranchNode.MATCH_FUL
L));
        logHangonStructOps(hangon,merged.getPairCount()-1);
```



```
      }
      if( mlistB != null ) {
        MergeEntry pair = mlistB.getEntry(mlistB.findPartner(me));
        if( pair != null && !checkHangonCombine(me,pair,mlistA,mlistB) ) {
          for(int ih=0;ih<pair.getHangonCount();ih++) {
            BranchNode hangon = pair.getHangon(ih).getNode();
            merged.append(hangon,hangon.getFirstPartner(BranchNode.MATCH_F
ULL));
            logHangonStructOps(hangon,merged.getPairCount()-1);
          }
        }
      }
    }
    return merged;
  }


  /** Determine and log operation on hangon. */
  protected void logHangonStructOps( BranchNode n, int childPos ) {
    if( !n.hasBaseMatch() )
      elog.insert(n,childPos);
    else if( (n.isLeftTree() && n.getBaseMatch().getLeft().getMatchCount() > 1 ) ||
          (!n.isLeftTree() && n.getBaseMatch().getRight().getMatchCount() > 1 ))
      elog.copy(n,childPos); // hangon with base match = copy
    else
      elog.move(n,childPos);
  }

  /** Determine and log operation on entry in merge list. */
  protected void logEntryStructOps(MergeEntry m1, MergeEntry m2, int childPo
s) {
    if( m1.moved )
      elog.move(m1.getNode(),childPos);
    else if( m2.moved )
      elog.move(m2.getNode(),childPos);
  }

  /** Log update operation */
  protected void logUpdateOperation( BranchNode n ) {
    elog.update(n);
  }

  /** Combine two merge lists into a merge pair list. The order of the merged
   *  children is decided here.
   */
  protected MergePairList makeMergePairList( MergeList mlistA, MergeList mlist
B ) {
    MergePairList merged = new MergePairList();
    removeDeletedOrMoved( mlistA, mlistB );
    elog.checkPoint();
    // Now we should have exactly the same entries in mlistA and mlistB
    // quick check
    if( mlistA.getEntryCount() != mlistB.getEntryCount() )
      throw new RuntimeException(
         "ASSERTION FAILED: MergeList.merge(): lists different lengths!");

    int posA = 0, posB = 0;
    MergeEntry ea = mlistA.getEntry(posA), eb= mlistB.getEntry(posB);
    while(true) {
      // Dump hangons from ea.. (we do this first as __START__ may have hangon
s)
      for(int i=0;i<ea.getHangonCount();i++) {
        BranchNode na = ea.getHangon(i).getNode();
        merged.append(na,na.getFirstPartner(BranchNode.MATCH_FULL));
        logHangonStructOps(na,merged.getPairCount()-1);
      }
      // And then from eb..
```



```
      if( eb.getHangonCount() > 0 ) {
        // Append b hangons (unless they were equal to the hangons of ea)
        if( !checkHangonCombine(ea,eb,mlistA,mlistB) ) {
          for(int i=0;i<eb.getHangonCount();i++) {
            BranchNode nb = eb.getHangon(i).getNode();
            merged.append(nb,nb.getFirstPartner(BranchNode.MATCH_FULL));
            logHangonStructOps(nb,merged.getPairCount()-1);
          }
        }
      }
      // end hangon dump
      int nextA=-1,nextB=-1;
      // figure out the next one
      nextA = ea.locked &&
              mlistA.getEntry(posA+1).locked ? posA + 1 : -1; // -1 means free
      nextB = eb.locked && mlistB.getEntry(posB+1).locked ? posB + 1 : -1;
      if( nextA == -1 && nextB == -1 ) { // No locking, just let both go forward
        nextA = posA + 1;
        nextB = posB + 1;
      }
      // Handle free positions
      if( nextB == -1 )
        nextB = mlistB.findPartner(mlistA.getEntry(nextA));
      else if (nextA == -1 )
        nextA = mlistA.findPartner(mlistB.getEntry(nextB));
      else if (nextB != mlistB.findPartner(mlistA.getEntry(nextA))) {
        // CONFLICTCODE
        clog.addListConflict( ConflictLog.MOVE,
        "Conflicting moves inside child list, using the sequencing of branch 1"
,
          ea.getNode() != START ? ea.getNode().getBaseMatch() : null,
          ea.getNode() != START ? ea.getNode() : null,
          eb.getNode() != START ? eb.getNode() : null );
        elog.rewind(); // Remove all edit ops made by this list merge attempt
        // Fallback on mergeListToPairList.
        return mergeListToPairList(mlistA.getEntryParent().isLeftTree() ? mlistA : m
listB,
          mlistA.getEntryParent().isLeftTree() ? mlistB : mlistA);
      }
      posA = nextA;
      posB = nextB;
      ea = mlistA.getEntry(posA);
      eb = mlistB.getEntry(posB);
      // See if we're done
      if( ea.node == END || eb.node == END ) {
        if( ea.node != eb.node )
          throw new RuntimeException(
          "ASSERTION FAILED: Merge.mergeLists(). Both cursors not at end");
        break;
      }
      // pos is set up so that ea and eb are merge-partners
      merged.append(ea.getNode(),eb.getNode());
      logEntryStructOps(ea,eb,merged.getPairCount()-1);
    }
    // Success, commit all edits generated to the log!
    elog.commit();
    return merged;
  }

  /** Check the situation of hangons that need to be combined, and generate
   *  conflict log entries.
   *  @return true if hangons are equal.
   */
  protected boolean checkHangonCombine(MergeEntry ea, MergeEntry eb,
                 MergeList mla, MergeList mlb ) {
    boolean hangonsAreEqual = false;
    if( ea.getHangonCount() > 0 ) {
```

```
      // Check if the hangons match _exactly_ (no inserts, and exactly same
      // sequence of copies). Then we may include the hangons just once. This
      // resembles the case when content of two nodes has been updated the sam
e
      // way... not a conflict, but maybe suspicious.
      // NOTE! We need to match the entire subtrees rooted at the hangon, that's
      // why treeMatches is used. Otherwise <p>Hello</p> and <p>World</p> woul
d
      // be considered equal (since <p>=<p>)
      if( eb.getHangonCount() == ea.getHangonCount() ) {
        hangonsAreEqual = true;
        for(int i=0;hangonsAreEqual && i<ea.getHangonCount();i++)
          hangonsAreEqual = treeMatches( eb.getHangon(i).getNode(),
                                         ea.getHangon(i).getNode() );
      }
      // Both have hangons, CONFLICTCODE
      // for now, chain A and B hangons
      if( hangonsAreEqual )
        // How should we encode the inserts, i.e. tell which nodes were inserted
        clog.addListWarning(ConflictLog.INSERT,
          "Equal insertions/copies in both branches after the context nodes.",
          ea.getNode().getBaseMatch() != null ? ea.getNode().getBaseMatch() :
          eb.getNode().getBaseMatch(), ea.getNode() != START ? ea.getNode() :
          null , eb.getNode() != START ? eb.getNode() : null );
      else
        clog.addListWarning(ConflictLog.INSERT, "Insertions/copies in both " +
          "branches after the context nodes. Sequencing the insertions.",
          ea.getNode().getBaseMatch() != null ? ea.getNode().getBaseMatch() :
          eb.getNode().getBaseMatch(),ea.getNode() != START ? ea.getNode() :
          null, eb.getNode() != START ? eb.getNode() : null );
    }
    return hangonsAreEqual;
  }

  /** NOP operation code. */
  protected static final int NOP = 1;
  /** MOVE_I operation code. Move inside childlist. */
  protected static final int MOVE_I = 2;
  /** MOVE_F operation code. Move outside childlist. */
  protected static final int MOVE_F = 3;
  /** DELETE operation code. */
  protected static final int DELETE = 4;

  /** Get operation on node in merge list.
   *  @return the operation on the node; one of NOP, MOVE_I, MOVE_F, DELET
E */
  protected int getOperation( BaseNode bn, MergeList ml ) {
    int mlPos = ml.matchInList(bn);
    if( mlPos == -1 ) {
      // Movef or delete
      MatchedNodes copiesInThisTree = null;
      if( ml.getEntryParent().isLeftTree() )
        copiesInThisTree = bn.getLeft();
      else
        copiesInThisTree = bn.getRight();
      if( copiesInThisTree.getMatches().isEmpty() )
        return DELETE;
      else
        return MOVE_F;
    } else {
      if( ml.getEntry(mlPos).isMoved() )
        return MOVE_I;
      else
        return NOP;
    }
  }
```



```
// $Id: Node.java,v 1.9 2001/09/05 13:21:26 ctl Exp $ D

import java.util.Vector;

/** Node in the parse tree. Each node in the parse trees has 0-n children,
 * content and a tag to identify nodes in the same matching subtrees (the
 * <code>matchArea</code> field). In addition, all nodes except the root
 * have a parent.
 */

public abstract class Node {

  protected Vector children = new Vector();
  protected XMLNode content = null;
  protected Node parent = null;
  protected int childPos=-1; // zero-based, i.e. first child = 0
  protected MatchArea area = null;

  public Node() {
    parent = null;
    childPos = -1;
  }

  public void addChild( Node n) {
    n.parent=this;
    n.childPos=children.size();
    children.add(n);
  }

  public Node getParentAsNode() {
    return parent;
  }

  public int getChildCount() {
    return children.size();
  }

  public Node getChildAsNode(int ix) {
    return (Node) children.elementAt(ix);
  }

  public boolean hasLeftSibling() {
    return childPos > 0;
  }

  public boolean hasRightSibling() {
    return parent != null && childPos < parent.children.size()-1;
  }

  public Node getLeftSibling() {
    if( parent == null || childPos == 0 )
      return null;
    else
      return parent.getChildAsNode(childPos-1);
  }

  public Node getRightSibling() {
    if( parent == null || childPos == parent.children.size() -1 )
      return null;
    else
      return parent.getChildAsNode(childPos+1);
  }

  public XMLNode getContent( ) {
    return content;
  }
```



```
  public int getChildPos() {
    return childPos;
  }

  public MatchArea getMatchArea() {
    return area;
  }

  public void setMatchArea(MatchArea anArea) {
    area=anArea;
  }
}
```

NodeFactory.java — Page 1 of 1

```java
// $Id: NodeFactory.java,v 1.6 2001/09/05 21:22:29 ctl Exp $ D

/** Node factory. Used to build trees, whose node type is not known at
 *  compile time.
 */
public abstract class NodeFactory {

  public NodeFactory() {
  }

  public abstract Node makeNode( XMLNode content );
}
```

ParseException.java — Page 1 of 1

```java
// $Id: ParseException.java,v 1.2 2001/09/05 21:22:29 ctl Exp $ D

/** Generic parse exception when reading XML. */
public class ParseException extends Exception {

  public ParseException( String message) {
    super(message);
  }
}
```

Patch.java — Page 1 of 3

```java
// $Id: Patch.java,v 1.5 2001/09/05 21:22:29 ctl Exp $ D

import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.HashSet;
import java.util.Vector;
import java.util.LinkedList;
import org.xml.sax.SAXException;
import org.xml.sax.ContentHandler;

/** Patching algorithm.
 *  See the {@link patch(BaseNode, BranchNode, org.xml.sax.ContentHandler)
 *  patch} method for usage.
 */

public class Patch {

  private static final Set RESERVED;
  private static final String DIFF_NS ="diff:";
  static {
      RESERVED = new HashSet();
      RESERVED.add(DIFF_NS+"copy");
      RESERVED.add(DIFF_NS+"insert");
      RESERVED.add(DIFF_NS+"esc");
  }

  public Patch() {
  }

  /** Patch a tree.
   *  @param base Tree to patch
   *  @param diff Parse tree of XML diff file, as produced by the Diff class
   *  @param ch Content handler that the patched tree is output to
   */

  public void patch( BaseNode base, BranchNode diff, ContentHandler ch )
    throws ParseException, SAXException {
    BranchNode patched = patch( base, diff );
    ch.startDocument();
    dumpTree( patched.getChild(0), ch );
    ch.endDocument();
  }

  private void dumpTree( BranchNode n, ContentHandler ch ) throws SAXExce
ption {
    if( n.getContent() instanceof XMLTextNode ) {
      char [] text = ((XMLTextNode) n.getContent()).getText();
      ch.characters(text,0,text.length);
    } else {
      XMLElementNode en = (XMLElementNode) n.getContent();
      ch.startElement("","",en.getQName(),en.getAttributes());
      for( int i=0;i<n.getChildCount();i++)
        dumpTree(n.getChild(i),ch );
      ch.endElement("","",en.getQName());
    }
  }

  protected BranchNode patch( BaseNode base, BranchNode diff ) throws
          ParseException  {
    initLookup(base,nodeLookup);
    BranchNode patch = new BranchNode( new XMLElementNode("$ROOT$",
      new org.xml.sax.helpers.AttributesImpl() ) );
      // Getchild(0) to skip diff tag
    copy( patch,diff.getChild(0),base.getChild(0));
    return patch;
  }
```

```
  // diff => a command or just some nodes to insert
  // patch = the parent node, under which the subtree produced by the command
  // shall be inserted

  protected void insert( BranchNode patch, BranchNode diff) throws
    ParseException {
    XMLNode cmdcontent = diff.getContent();
    if( cmdcontent instanceof XMLTextNode ||
      !RESERVED.contains(((XMLElementNode) cmdcontent).getQName())) {
      // Simple insert operation
      BranchNode node = new BranchNode( cmdcontent );
      patch.addChild(node);
      for( int i=0;i<diff.getChildCount();i++)
        insert(node,diff.getChild(i)); // Recurse to next level
    } else {
      // Other ops..
      XMLElementNode ce = (XMLElementNode) cmdcontent;
      if( ce.getQName().equals(DIFF_NS+"esc") ||
          ce.getQName().equals(DIFF_NS+"insert") ) {
        if( diff.getChildCount() == 0)
          throw new ParseException("DIFFSYNTAX: insert/esc has no subtree
" +
                                   ce.toString() );
        insert( patch, diff.getChild(0) );
      } else {
        // Copy operation
        BaseNode srcRoot = null;
        try {
          srcRoot = locateNode(
                    Integer.parseInt(ce.getAttributes().getValue("src"))) ;
        } catch ( Exception e ) {
          throw new ParseException(
            "DIFFSYNTAX: Invalid parameters in command " + ce.toString() );
        }
        copy( patch, diff, srcRoot );
      } // Copyop
    }
  }

  protected void copy( BranchNode patch, BranchNode diff, BaseNode srcRoot
)
                   throws ParseException {
    // Gather the stopnodes for the copy
    Vector dstNodes = new Vector();
    Map stopNodes = new HashMap();
    for( int i = 0; i < diff.getChildCount(); i++ ) {
      XMLNode content = diff.getChild(i).getContent();
      Node stopNode = null;
      // Sanity check
      if( content instanceof XMLTextNode || (
        !((XMLElementNode) content).getQName().equals(DIFF_NS+"copy") &&
        !((XMLElementNode) content).getQName().equals(DIFF_NS+"insert") ) )
        throw new ParseException("DIFFSYNTAX: Only copy or insert comman
ds may"+
                              " appear below a copy command");
      XMLElementNode stopCommand = (XMLElementNode) content;
      try {
        // PATCH here when implementing RUN attribute...
        stopNode = locateNode( Integer.parseInt(
                        stopCommand.getAttributes().getValue("dst")));
      } catch ( Exception e ) {
        throw new ParseException("DIFFSYNTAX: Invalid parameters in comma
nd " +
                        stopNode.toString() );
      }
      dstNodes.add( stopNode );
```



```
      if( !stopNodes.containsKey(stopNode) )
        stopNodes.put(stopNode,null);
    }
    // Run copy. stopNode values are at the same time filled in to point to the
    // created nodes
    dfsCopy( patch, srcRoot , stopNodes );
    // Recurse for each diff child
    for( int i = 0; i < diff.getChildCount(); i++ ) {
      insert( (BranchNode) stopNodes.get( dstNodes.elementAt(i) ),
              diff.getChild(i));
    }
  }

  protected void dfsCopy( BranchNode dst, BaseNode src, Map stopNodes ) {
    BranchNode copied = new BranchNode( src.getContent() );
    dst.addChild( copied );
    if( stopNodes.containsKey(src) ) {
      stopNodes.put(src,copied);
      return; // We're done in this branch
    }
    for( int i =0;i<src.getChildCount();i++)
      dfsCopy( copied, src.getChild(i), stopNodes );
  }

  private Vector nodeLookup = new Vector();

  protected BaseNode locateNode( int ix ) {
    return (BaseNode) nodeLookup.elementAt(ix);
  }

  // BFS Enumeration of nodes. Useful beacuse adjacent nodes have subsequent
ids =>
  // diff can use the "run" attribute more often
  // NOTE: Copied from diff, should really be moved to a common base class for
  // patch & diff
  protected void initLookup( Node start, Vector table) {
    LinkedList queue = new LinkedList();
    queue.add(start);
    while( !queue.isEmpty() ) {
      Node n = (Node) queue.removeFirst();
      table.add(n);
      for( int i=0;i<n.getChildCount();i++)
        queue.add(n.getChildAsNode(i));
    }
  }
}
```



```
// $Id: PathTracker.java,v 1.2 2001/09/05 21:22:29 ctl Exp $ D

import java.util.LinkedList;
import java.util.Iterator;
import java.util.Collections;

/** Tracks current position in a tree. */

public class PathTracker {

  static final char PATHSEP='/';

  private LinkedList path = null;
  private int childPos = -1;

  /** Create new tracker. The location is initialized to the root, or "/" */
  public PathTracker() {
    resetContext();
  }

  /** The location is initialized to the root, or "/" */
  public void resetContext() {
    path = new LinkedList();
    childPos = 0;
  }

  /** Move to next child. Must be in a subtree, created by enterSubtree() */
  public void nextChild() {
    childPos++;
  }

  /** Start new subtree */
  public void enterSubtree() {
    path.addLast(new Integer(childPos));
    childPos = 0;
  }

  /** End subtree. */
  public void exitSubtree() {
    Integer oldpos = (Integer) path.removeLast();
    childPos = oldpos.intValue();
  }

  /** Get string describing current location, excluding current child position */
  public String getPathString() {
    return getPathString(path,-1,false);
  }

  /** Get string describing current location, including current child position */
  public String getFullPathString() {
    return getPathString(path,childPos,true);
  }

  /** Get string describing current location, including current child position */
  public String getPathString(int achildPos) {
    return getPathString(path,achildPos,true);
  }

  /** Get string describing current location of a node in a tree. Exclude the
   *  child position of the node (path ends in "/"). */
  public static String getPathString(Node n ) {
    return getPathString(makePath(n),-1,false);
  }

  /** Get string describing current location of a node in a tree. Include the
   *  child position of the node (path ends in a number). */
  public static String getPathString(Node n,int achildPos) {
```

```java
    return getPathString(makePath(n),achildPos,true);
  }

  // Create path string from linked list of nodes.
  private static String getPathString( LinkedList path, int childPos,
    boolean useChildPos ) {
    StringBuffer p = new StringBuffer();
    Iterator i=path.iterator(); // Skip artificial root node
    i.next();
    for(;i.hasNext();) {
      p.append(PATHSEP);
      p.append(((Integer) i.next()).toString());
    }
    if( useChildPos ) {
      p.append(PATHSEP);
      p.append(childPos);
    }
    return p.toString();
  }

  // Create linked list of nodes to the root from a node in a tree.
  private static LinkedList makePath( Node n ) {
    LinkedList path = new LinkedList();
    do {
      path.addLast(new Integer(n.getChildPos()));
    } while( (n = n.getParentAsNode()) != null);
    Collections.reverse(path);
    return path;
  }
}
```



```java
// $Id: TreeDiffMerge.java,v 1.4 2001/09/05 21:22:30 ctl Exp $ D
import gnu.getopt.*;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.PrintStream;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.Attributes;

/** Driver class for 3DM. Parses command line and runs the merge/diff/patch
 *  algorithms.
 */
public class TreeDiffMerge {
  public static void main(String[] args) throws java.io.IOException {
    System.err.println(
    "3DM XML Tree Differencing and Merging Tool. PROTOTYPE: $Revision:
1.4 $" );
    // Get command line options
    int firstFileIx = parseOpts( args );
    if( op == MERGE && (args.length - firstFileIx) == 3 )
      merge( firstFileIx, args, System.out );
    else if( op == MERGE && (args.length - firstFileIx) == 4 )
      merge( firstFileIx, args, new FileOutputStream(args[firstFileIx+3]));
    else if( op == DIFF && (args.length - firstFileIx) == 2 )
      diff( firstFileIx, args, System.out );
    else if( op == DIFF && (args.length - firstFileIx) == 3 )
      diff( firstFileIx, args, new FileOutputStream(args[firstFileIx+2]));
    else if( op == PATCH && (args.length - firstFileIx) == 2 )
      patch( firstFileIx, args, System.out );
    else if( op == PATCH && (args.length - firstFileIx) == 3 )
      patch( firstFileIx, args, new FileOutputStream(args[firstFileIx+2]));
    else {
      System.err.println("Usage: 3dm [options] {-m base branch1 branch2|-d "+
      "base branch1 |-p base patch} [outfile]" );
      System.err.println("Use the -m (or --merge) option to merge the files "+
      "base, branch1 and branch2"  );
      System.err.println("Use the -d (or --diff) option to diff the files "+
      "base and branch1"  );
      System.err.println("Use the -p (or --patch) option to patch the file "+
      "base with the file patch"  );
      System.err.println("The options are:");
      System.err.println("-e, --editlog[=logfile]");
      System.err.println("   Log edit operations to logfile, default edit.log");
      System.err.println("-c, --copythreshold=bytes");
      System.err.println("   Threshold for considering a duplicate structure "+
                         "to be a copy. Default value is " +
                         Matching.COPY_THRESHOLD + " bytes");
    }
  }

  // Factory for BaseNode:s
  private static NodeFactory baseNodeFactory =  new NodeFactory() {
          public Node makeNode(  XMLNode content ) {
            return new BaseNode( content  );
          }
      };

  // Factory for BranchNode:s
  private static NodeFactory branchNodeFactory =  new NodeFactory() {
          public Node makeNode(  XMLNode content ) {
            return new BranchNode(  content  );
          }
      };

  /** Runs merge algorithm.
   *  @param ix Index in args of first file
   *  @param out Outputstream for merged tree */
```



```java
  protected static void merge( int ix, String[] args, OutputStream out ) {
    BaseNode docBase=null;
    BranchNode docA=null,docB=null;
    String currentFile = "";
    try {
      XMLParser p = new XMLParser();
      currentFile = args[ix+0];
      docBase = (BaseNode) p.parse( currentFile,baseNodeFactory);
      currentFile = args[ix+1];
      docA = (BranchNode) p.parse( currentFile, branchNodeFactory);
      currentFile = args[ix+2];
      docB = (BranchNode) p.parse( currentFile,branchNodeFactory);
    } catch ( Exception e ) {
      System.err.println("XML Parse error in " + currentFile +
                         ". Detailed exception info is:" );
      System.err.println( e.toString() );
      e.printStackTrace();
      return;
    }
    try {
      Merge merge = new Merge( new TriMatching( docA, docBase, docB ) );
      merge.merge( new XMLPrinter( new PrintWriter( out)  ) );
      merge.getConflictLog().writeConflicts(new XMLPrinter(
        new PrintWriter( new FileOutputStream( conflictLogName ))));
      if( editLog )
        merge.getEditLog().writeEdits( new XMLPrinter(
          new PrintWriter( new FileOutputStream( editLogName ))));
    } catch ( Exception e ) {
      System.err.println("Exception while merging.. trace follows:");
      System.err.println( e.toString() );
      e.printStackTrace();
    }
  }

  /** Runs diff algorithm.
   *  @param ix Index in args of first file
   *  @param out Outputstream for merged tree */

  protected static  void diff( int ix, String[] args, OutputStream out ) {
    BaseNode docBase=null;
    BranchNode docA=null;
    String currentFile = "";
    try {
      XMLParser p = new XMLParser();
      currentFile = args[ix+0];
      docBase = (BaseNode) p.parse( currentFile,baseNodeFactory);
      currentFile = args[ix+1];
      docA = (BranchNode) p.parse( currentFile, branchNodeFactory);
    } catch ( Exception e ) {
      System.err.println("XML Parse error in " + currentFile +
                         ". Detailed exception info is:" );
      System.err.println( e.toString() );
      e.printStackTrace();
      return;
    }
    try {
      Matching m = new DiffMatching( docBase, docA );
      Diff diff = new Diff( m );
      diff.diff(new XMLPrinter( new PrintWriter(out) ));
    } catch ( Exception e ) {
      System.err.println("Exception while diffing.. trace follows:");
      System.err.println( e.toString() );
      e.printStackTrace();
    }
  }
  /** Runs patch algorithm.
   *  @param ix Index in args of first file
```

TreeDiffMerge.java Page 3 of 4

```
   * @param out Outputstream for merged tree */
  static  void patch( int ix, String[] args, OutputStream out ) {
    BaseNode docBase=null;
    BranchNode docPatch=null;
    String currentFile = "";
    try {
      XMLParser p = new XMLParser();
      currentFile = args[ix+0];
      docBase = (BaseNode) p.parse( currentFile,baseNodeFactory);
      currentFile = args[ix+1];
      docPatch = (BranchNode) p.parse( currentFile, branchNodeFactory);
    } catch ( Exception e ) {
      System.err.println("XML Parse error in " + currentFile +
                         ". Detailed exception info is:" );
      System.err.println( e.toString() );
      e.printStackTrace();
      return;
    }
    try {
      Patch patch = new Patch();
      patch.patch(docBase,docPatch, new XMLPrinter( new PrintWriter( out  ) ) );
    } catch ( Exception e ) {
      System.err.println("Exception while diffing.. trace follows:");
      System.err.println( e.toString() );
      e.printStackTrace();
    }
  }

  // Default files
  public static final String EDITLOG = "edit.log";
  public static final String CONFLICTLOG = "conflict.log";
  // operation codes, returned by parseOpts()
  public static final int MERGE = 0;
  public static final int DIFF = 1;
  public static final int PATCH = 2;

  protected static boolean editLog = false;
  protected static String editLogName = EDITLOG;
  protected static String conflictLogName = CONFLICTLOG;
  protected static int op = -1;

  // Parse command line options.
  private static int parseOpts( String args[] ) {
    LongOpt lopts[] = {
      new LongOpt("editlog",LongOpt.OPTIONAL_ARGUMENT,null,'e'),
      new LongOpt("copythreshold",LongOpt.REQUIRED_ARGUMENT,null,'c'),
      new LongOpt("merge",LongOpt.NO_ARGUMENT,null,'m'),
      new LongOpt("diff",LongOpt.NO_ARGUMENT,null,'d'),
      new LongOpt("patch",LongOpt.NO_ARGUMENT,null,'p'),
    };
    Getopt g = new Getopt("3DM", args, "e::c:mdp", lopts);
    int c;
    String arg;
    while ((c = g.getopt()) != -1) {
     switch(c) {
         case 'e':
           editLog = true;
           editLogName = getStringArg(g,EDITLOG);
           break;
         case 'c':
           Matching.COPY_THRESHOLD = getIntArg(g,Matching.COPY_THRESH
OLD);
           break;
         case 'm':
           op = MERGE;
           break;
         case 'p':
```

TreeDiffMerge.java Page 4 of 4

```
           op = PATCH;
           break;
         case 'd':
           op = DIFF;
           break;
     }
    }
    return g.getOptind();
  }

  static String getStringArg( Getopt g, String defval ) {
    String arg = g.getOptarg();
    if( arg == null || "?".equals(arg) )
      return defval;
    else
      return arg;
  }

  static int getIntArg( Getopt g, int defval ) {
    String arg = g.getOptarg();
    if( arg == null || "?".equals(arg) )
      return defval;
    else {
      try {
        return Integer.parseInt(arg);
      } catch (Exception e) {
        return defval;
      }
    }
  }

}
```

TriMatching.java Page 1 of 1

```
// $Id: TriMatching.java,v 1.11 2001/09/05 21:22:30 ctl Exp $ D

/** Matching between a base and two branch trees. */
public class TriMatching extends Matching {

  private BranchNode leftRoot = null;
  private BranchNode rightRoot = null;

  /** Create matching */
  public TriMatching( BranchNode left, BaseNode base, BranchNode right ) {
    super( base, right );
    leftRoot =left;
    rightRoot = right;
    swapLeftRight( base );
    buildMatching( base, left );
    setPartners( left, false );
    setPartners( right, true );
  }


  // Swap left and right matching fields in base nodes. The superclass
  // always fills in left matchings, so we need to call this when making
  // the right (no pun intended) matchings
  protected void swapLeftRight( BaseNode base ) {
    base.swapLeftRightMatchings();
    for( int i=0;i<base.getChildCount();i++)
      swapLeftRight(base.getChild(i));
  }

  // Set partner fields of branch nodes
  protected void setPartners( BranchNode n, boolean partnerInLeft ) {
    BaseNode baseMatch = n.getBaseMatch();
    if( baseMatch != null ) {
      if( partnerInLeft )
        n.setPartners(baseMatch.getLeft());
      else
        n.setPartners(baseMatch.getRight());
    } else
        n.setPartners(null);
    for( int i=0;i<n.getChildCount();i++)
      setPartners(n.getChild(i),partnerInLeft);
  }

  public BranchNode getLeftRoot() {
    return leftRoot;
  }

  public BranchNode getRightRoot() {
    return rightRoot;
  }
}
```

XMLElementNode.java Page 1 of 2

```
// $Id: XMLElementNode.java,v 1.8 2001/09/05 21:22:30 ctl Exp $ D

import org.xml.sax.Attributes;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Map;
import org.xml.sax.helpers.AttributesImpl;
import org.xml.sax.Attributes;
import java.security.MessageDigest;

/** Stores XML element nodes. */
public class XMLElementNode extends XMLNode {

  private String name = null;
  private AttributesImpl attributes = null;
  private int nHashCode = -1;
  private byte[] attrHash = null;

  public XMLElementNode( String aname, Attributes attr ) {
    name = aname;
    attributes = new AttributesImpl( attr );
    makeHash();
  }

  private void makeHash() {
    nHashCode = name.hashCode();
    infoSize = Measure.ELEMENT_NAME_INFO;
    MessageDigest md = getMD();
    for( int i=0;i<attributes.getLength();i++) {
      int vsize = attributes.getValue(i).length();
      infoSize += Measure.ATTR_INFO + (vsize > Measure.ATTR_VALUE_THRE
SHOLD ? vsize -
         Measure.ATTR_VALUE_THRESHOLD : 1 );
      md.update( calculateHash( attributes.getQName(i) ) );
      md.update( calculateHash( attributes.getValue(i) ) );
    }
    attrHash = md.digest();
  }

  /** DUMMY! Always returns "" */
  public String getNamespaceURI() {
    return "";
  }

  /** DUMMY! Always returns "" */
  public String getLocalName() {
    return "";
  }

  public String getQName() {
    return name;
  }

  public Attributes getAttributes() {
    return attributes;
  }

  public void setAttributes(Attributes atts) {
    attributes=new AttributesImpl( atts );
  }

  public String toString() {
    StringBuffer sb = new StringBuffer();
    sb.append(name);
    sb.append(" {");
    if( attributes != null && attributes.getLength() > 0) {
      for( int i = 0;i<attributes.getLength();i++) {
```

XMLElementNode.java Page 2 of 2

```
        sb.append(' ');
        sb.append(attributes.getQName(i) );
        sb.append('=');
        sb.append(attributes.getValue(i));
      }

    }
    sb.append('}');
    return sb.toString();
  }

  public boolean contentEquals( Object a ) {
    if( a instanceof XMLElementNode )
      return ((XMLElementNode) a).nHashCode == nHashCode &&
       MessageDigest.isEqual(((XMLElementNode) a).attrHash,attrHash);
    else
      return false;
  }


  public int getContentHash() {
  return (attrHash[0]+attrHash[1]<<8+attrHash[2]<<16+attrHash[3]<<24)^nHash
Code;
  }
}
```

XMLNode.java Page 1 of 1

```
// $Id: XMLNode.java,v 1.7 2001/09/05 21:22:30 ctl Exp $ D

import java.security.MessageDigest;

/** Class for storing content of XML nodes. Supports fast equality comparison
 *  using MD5 hash codes, and automatic calculation of node infoSize. */

public abstract class XMLNode {

  protected int infoSize = 0;

  public XMLNode() {
  }

  public int getInfoSize() {
    return infoSize;
  }

  public abstract boolean contentEquals( Object a );
  /** Get 32-bit hash code */
  public abstract int getContentHash();

  protected MessageDigest getMD() {
    try{
      return MessageDigest.getInstance("MD5");
    } catch ( java.security.NoSuchAlgorithmException e ) {
      System.err.println("MD5 hash generation not supported -- aborting");
      System.exit(-1);
    }
    return null;
  }

  protected byte[] calculateHash(char[] data) {
    MessageDigest contentHash = getMD();
   contentHash.reset();
    for( int i=0;i<data.length;i++) {
      contentHash.update((byte) (data[i]&0xff));
      contentHash.update((byte) (data[i]>>8));
    }
    return contentHash.digest();
  }

  protected byte[] calculateHash(String data) {
    MessageDigest contentHash = getMD();
    contentHash.reset();
    for( int i=0;i<data.length();i++) {
      contentHash.update((byte) (data.charAt(i)&0xff));
      contentHash.update((byte) (data.charAt(i)>>8));
    }
    return contentHash.digest();
  }
}
```

XMLParser.java Page 1 of 2

```java
// $Id: XMLParser.java,v 1.4 2001/09/05 21:22:30 ctl Exp $ D

import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.AttributesImpl;
import org.xml.sax.InputSource;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXException;
import java.util.Stack;
import java.io.FileReader;
import java.io.FileNotFoundException;

/** 3DM wrapper for a generic XML SAX parser. */

public class XMLParser extends DefaultHandler {

  /** Default parser name. */
  private static final String DEFAULT_PARSER_NAME = "org.apache.xerces.
parsers.SAXParser";
  private  XMLReader xr = null;

  public XMLParser() throws Exception {
    this(DEFAULT_PARSER_NAME);
  }

  /** Create new parser using supplied SAX parser class name. */
  public XMLParser( String saxParserName ) throws Exception {
    try {
      xr = (XMLReader)Class.forName(saxParserName).newInstance();
    } catch (Exception e ) {
      new Exception("Unable to instantiate parser " + saxParserName );
    }
    xr.setContentHandler(this);
    xr.setErrorHandler(this);
    try {
     xr.setFeature("http://xml.org/sax/features/namespaces",false);
     xr.setFeature("http://xml.org/sax/features/validation",false);
    } catch (SAXException e) {
     throw new Exception("Error setting features:" + e.getMessage());
    }

  }

  // Parser state
  private String currentText = null;
  private Node currentNode = null;
  private NodeFactory factory = null;
  private Stack treestack = new Stack();


  /** Parse an XML file. Returns a parse tree of the XML file.
   *  @param file Input XML file
   *  @param aFactory Factory for creating nodes in the tree.
   */
  public Node parse( String file, NodeFactory aFactory ) throws ParseException,
         java.io.FileNotFoundException, java.io.IOException {
    factory = aFactory;
    FileReader r = new FileReader(file);
    try {
      xr.parse(new InputSource(r));
    } catch ( org.xml.sax.SAXException x ) {
      throw new ParseException(x.getMessage());
    }
    Node root = currentNode;
    // Don't leave a ptr to the parsed tree; it can't be GC'd then!
    currentNode = null;
```

XMLParser.java Page 2 of 2

```java
    factory = null; // forget factory and allow GC
    return root;
  }

  public void startDocument () {
    currentNode = factory.makeNode( new XMLElementNode("$ROOT$",
      new AttributesImpl() ) );
  }

  public void endDocument () {
  }

  public void startElement (String uri, String name,
                            String qName, Attributes atts) {
    if( currentText != null )
      currentNode.addChild( factory.makeNode(
       new XMLTextNode( currentText.trim().toCharArray() )  ) );
    currentText = null;
    Node n = factory.makeNode( new XMLElementNode( qName, atts ) );
    currentNode.addChild(  n  );
    treestack.push( currentNode );
    currentNode = n;
  }


  public void endElement (String uri, String name, String qName)
  {
    if( currentText != null )
      currentNode.addChild( factory.makeNode(
          new XMLTextNode( currentText.trim().toCharArray() ) ) );
    currentText = null;
    currentNode = (Node) treestack.pop();
  }


  public void characters (char ch[], int start, int length)
  {
    // The method trims whitespace from start and end of character data
    boolean lastIsWS = currentText == null || currentText.endsWith(" ");
    StringBuffer sb = new StringBuffer();
    for( int i=start;i<start+length;i++) {
      if( Character.isWhitespace(ch[i]) ){
        if( lastIsWS )
          continue;
        sb.append(" ");
        lastIsWS = true;
      } else {
        sb.append(ch[i]);
        lastIsWS = false;
      }
    }
    String chars = sb.toString(); //.trim();
    if( chars.trim().length() == 0 )
      return;
    if( currentText != null )
      currentText += chars;
    else {
      currentText = chars;
    }
  }
}
```

XMLPrinter.java Page 1 of 2

```java
// $Id: XMLPrinter.java,v 1.2 2001/06/20 13:25:59 ctl Exp $

import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.Attributes;

class XMLPrinter extends DefaultHandler {


  int indent = 0;
  private static final String IND =
  "                                                                    ";
  private java.io.PrintWriter pw = null;

  XMLPrinter( java.io.PrintWriter apw ) {
    pw=apw;
  }
   /////////////////////////////////////////////////////////////////////////
   // Event handlers.
   /////////////////////////////////////////////////////////////////////////


   public void startDocument ()
   {
     childcounter =HAS_CONTENT;
     pw.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
   }


   public void endDocument ()
   {
      //System.out.println("End document");
     pw.flush();
   }

    java.util.Stack csstack = new java.util.Stack();
   Integer childcounter = null;
   public void startElement (String uri, String name,
                        String qName, Attributes atts)

   {
    if( childcounter == null ) {
      pw.println(">");
     childcounter =HAS_CONTENT;
    }
    StringBuffer tagopen = new StringBuffer();
    tagopen.append('<');
    tagopen.append( qName );
//    tagopen.append(' ');
    if( atts != null && atts.getLength() != 0 ) {
      for( int i = 0;i<atts.getLength();i++ ) {
        tagopen.append(' ');
        tagopen.append(atts.getQName(i));
        tagopen.append('=');
        tagopen.append('"');
        tagopen.append(atts.getValue(i));
        tagopen.append('"');
      }
    }
    csstack.push( childcounter );
    childcounter = null;
//      if( assumeNoChildren )
//        tagopen.append("/>");
//      else
//        tagopen.append('>');
    pw.print(IND.substring(0,indent)  + tagopen.toString());
    indent ++;
   }
```

XMLPrinter.java — Page 2 of 2

```java
  public void endElement (String uri, String name, String qName)
  {
    indent--;
      if( childcounter == null )
        pw.println(" />");
      else
        pw.println(IND.substring(0,indent)+ "</"+qName+">");
    childcounter = (Integer) csstack.pop();
  }

  final Integer HAS_CONTENT = new Integer(0);

  public void characters (char ch[], int startpos, int length)
  {
    if(childcounter!=HAS_CONTENT)
      pw.println(">");
    childcounter = HAS_CONTENT;
    StringBuffer sb = new StringBuffer();
    for(int i=startpos;i<startpos+length;i++) {
      switch( ch[i] ) {
        case '<': sb.append("&lt;");
              break;
        case '>': sb.append(">");
              break;
        case '\'': sb.append("&apos;");
              break;
        case '&': sb.append("&amp;");
              break;
        case '"': sb.append("&quot;");
              break;
        default:
            sb.append(ch[i]);
      }
    }
    String chars = sb.toString();
//      String chars = new String( ch, startpos, length ).trim();
    if( chars.length() == 0 )
      return;/*
    int start=0,next=-1;
    do {
      next=chars.indexOf("\n",start);
      if( next==-1)
        pw.println(chars.substring(start));
      else {
        pw.println(chars.substring(start,next));
        start=next+1;
      }
    } while( next != -1 );*/
    pw.println(chars);
    //System.err.println("OUT:"+chars);
  }


}
```

XMLTextNode.java — Page 1 of 1

```java
// $Id: XMLTextNode.java,v 1.8 2001/09/05 21:22:30 ctl Exp $

import java.security.MessageDigest;

/** Stores XML element nodes. */
public class XMLTextNode extends XMLNode {

  private char[] text=null;
  private byte[] cHash = null;

  XMLTextNode( String srctext ) {
    this( srctext.toCharArray() );
  }

  XMLTextNode( char[] srctext ) {
    this( srctext,0,srctext.length);
  }

  XMLTextNode( char[] srctext, int first, int length ) {
    text = new char[length];
    System.arraycopy(srctext,first,text,0,length);
    cHash = calculateHash(text);
    infoSize = text.length > Measure.TEXT_THRESHOLD ? text.length -
          Measure.TEXT_THRESHOLD : 1;
  }

  public boolean contentEquals( Object a ) {
    if( a instanceof XMLTextNode )
      return MessageDigest.isEqual(cHash,((XMLTextNode) a).cHash);
    else
      return false;
  }

  public char[] getText() {
    return text;
  }

  public void setText(char[] aText) {
    text = aText;
  }

  public String toString() {
    return new String(text);
  }

  public int getContentHash() {
    return cHash[0]+cHash[1]<<8+cHash[2]<<16+cHash[3]<<24;
  }

}
```

Index

ancestor, 4
atomic data, 22
augmented merge tree, 92

BFS, 5
branch, 3
branch point, 3
breadth-first search, 5

children, 4
child list, 4
content distance, 61
content matched nodes, 49
content partner, 52
content string, 62
context awareness, 24
copied node, 53
coupling, 23
CVS, 31

data locking, 22
data synchronization, 21
decoupled, 23
deleted node, 52
delete conflict, 55
delete tree, 81
depth, 4
depth-first search, 5
depth level, 4
descendants, 4
DFS, 5
diff, 3
differencing, 3
differential transfer, 23

edge, 4
edit distance, 37
edit script, 37
edit scrip cost, 37
element, 6
element nodes, 7

far moved, 53
forest, 4
fully matched nodes, 49

hangon node, 77
height, 4

information content, 65
inserted node, 52
interior node, 4
in sequence, 53

labeled, 4
leaf node, 4
left locked, 79
left node context, 54
left sibling, 4
length, 4
linear data, 22
list partner, 53

matched nodes, 48
matching, 47
matching, in general, 37
matching subtree, 64
match string, 62
match type, 49
merge, 3
merge pair, 75
merge pair list, 75
merged child list, 55
merged node, 54
merge list, 77
meta language, 6
mobility, 25
move/move conflict, 55
move/update case, 30
moved node, 53

natural matching, 50
node, 4
node content, 5
node context, 54

optimistic locking, 23
ordered, 4

pair predecessor, 89
pair successor, 89
parent, 4
partner, 52
patching, 3
path, 4
PDA, 1
peer-to-peer, 25
persistent naming problem, 27
primary copy, 53

q-gram, 60
q-gram distance, 60
q-gram profile, 60
q-gram string distance, 60

revision tree, 3
revisions, 2
revison number, 2
right locked, 79
right node context, 54
right sibling, 4
root, 4
rooted tree, 4

secondary copies, 53
sequence number, 53
sequencing conflict, 56
services, 24
service discovery, 24
service interaction, 25
structurally matched nodes, 49
structural 3-way merge, 27
structural data, 22
structural partner, 52
subtree, 4
SVG, 14
synchronization granularity, 22
synchronization topology, 23
SyncML, 32

tag, 6
text nodes, 7
tree, 4
tree cursor, 75
tree differencing problem, 28
tree merge problem, 28
tree patching problem, 28
truncated subtree, 63

ubiquitous computing, 23
universal data access, 25
unordered, 4
update/update conflict, 55
updated node, 53

version, 3

WebDAV, 33
well-formed matching, 51

XHTML, 7
XML, 6
XPath, 7