

XML Three-way Merge as a Reconciliation Engine for Mobile Data

Tancred Lindholm
Helsinki Institute for Information Technology
P.O. Box 9800
FIN-02015 HUT, Finland
ctl@cs.hut.fi

ABSTRACT

Optimistic replication approaches are often employed on mobile devices, which raises the need for reconciliation of concurrently modified data. We propose that three-way merging algorithms, in particular those that are able to process tree-structured data in XML format, make good candidates for a generic data reconciliation engine on mobile devices.

By exchanging data through XML files we impose minimal constraints on application design and are able to offer reconciliation services to a large number of existing applications. Reconciliation support can be added to an application in several increments, allowing application developers to choose a suitable level of support compared to implementation effort. We give two examples of reconciliation by three-way merging of XML.

Categories and Subject Descriptors

C.2.4 [Computer Communication Networks]: Distributed Systems

General Terms

Design

Keywords

Reconciliation, optimistic replication, XML, three-way merge

1. INTRODUCTION

In the mobile environment we frequently face the challenges of weak connectivity, i.e. low bandwidth, high-latency links, and periods of disconnection. To allow sharing of data objects between devices in such an environment, a strategy of *optimistic replication* [11] of data is often employed. Optimistic replication strategies allow shared objects to be modified during periods of disconnection, which raises the

possibility that several independently modified replicas of an object may emerge. As we typically want each object to have an unambiguous content, we need to combine the contents of the modified replicas into a single unified content. We refer to this task as *data reconciliation* (also known as *data integration* or *merging*).

The ability to perform automatic reconciliation is particularly important in the mobile environment, as many applications become “involuntarily collaborative”. For instance, consider a simple text editor: in the desktop world, authors may take turns editing a document. In the mobile world, on the other hand, one of the authors may be disconnected from the network for an extended amount of time, making the approach of taking turns infeasible and raising the need for disconnected collaboration.

A solution is to add data reconciliation capabilities separately to each application. Unfortunately, this will complicate application development, making it less attractive to write mobile applications. Applications will also be larger, wasting the already limited resources of the mobile device.

There is thus a need for general data reconciliation facilities that incur little overhead on application development and on the size of the executable. In this paper we claim that generic three-way merging tools for XML files have several features that make them good candidates for use as mobile reconciliation engines. The scope of the paper is limited to applications that store data in files, providing us with clearly identifiable and easily accessible units of data.

The paper is organized as follows: In section 2, we introduce three-way merging as a concept as well as the particular XML three-way merge tool used in our research. In section 3 we argue for the suitability of XML three-way merging as a general reconciliation facility in the mobile environment. Two examples are presented in section 4. Following the examples, we give an overview of related work and our conclusions. The research presented in this paper originates in the Fuego Core project on Mobile Middleware (<http://www.hiit.fi/fuego>).

2. THREE-WAY MERGING OF XML

Assume that there exist two identical replicas of an object T_0 . The replicas of T_0 are independently edited, creating the objects T_1 and T_2 . The *edit scripts* (see e.g. [2]) \mathcal{E}_1 and \mathcal{E}_2 are the ordered sequences of edit operations that created T_1 and T_2 from T_0 . We wish to obtain a version of the object that integrates the modifications made to both replicas, i.e. we want to *reconcile* the changes made to the replicas into

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MobiDE'03, September 19, 2003, San Diego, California, USA.
Copyright 2003 ACM 1-58113-767-2/03/0009 ...\$5.00.

a unified version.

Reconciliation can be performed by a procedure known as three-way merging if we assume that T_0 is available when T_1 and T_2 are reconciled. We informally define the three-way merge [8] of the objects T_0 , T_1 , and T_2 to be the object T_m , which is obtained by applying the changes between T_0 and T_1 as well as the changes between T_0 and T_2 to T_0 . We refer to T_0 as the *base version*, T_1 and T_2 as the *changed versions*, and T_m as the *unified version*. Note that three-way merging, as defined here, does not require any knowledge of the edit scripts \mathcal{E}_1 and \mathcal{E}_2 . Furthermore, we limit the discussion to the reconciliation of two changed versions, but note that one should be able to reconcile an arbitrary number of replicas by performing successive three-way merges.

We may identify two phases of the three-way merge process: *change detection* and *reconciliation*. In the change detection phase the changes between T_0 and T_1 as well as those between T_0 and T_2 are identified. These changes are subsequently applied to T_0 during the reconciliation phase. If the edit scripts \mathcal{E}_1 and \mathcal{E}_2 are known, these may be used as a more accurate alternative to the change detection phase.

An important aspect of a three-way merging algorithm (TMA) is the data structure it is designed to operate upon. These include sets of tuples (for relational databases), ordered lists (for text files), trees and graphs (for structured data), as well as application specific structures. As data structures become more complex, the number of ways of changing the structures and integrating the changes increases, adding to the complexity of the TMA. In addition, the semantics of the data structures may affect how the merging should be performed. To deal with these challenges, merging tools for complex data are usually designed for the data format of a particular application.

Despite these difficulties, general TMAs for complex data structures are clearly useful. Frequently, an understanding of the full semantics of the data is not required to perform a successful merge. For instance, consider three-way merging of source code using the Unix `diff3` tool: the semantic structure of the source code may be very complex, yet we often obtain valid results from `diff3`, which considers the structure of its input data to be an ordered list of text lines.

In this paper we consider TMAs that operate on trees. Our reference implementation is the `3dm` XML differencing and three-way merging tool [8], which we think strikes a good balance between computational efficiency and complexity of tree edit operations that can be reconciled. In addition to updates, insertions, and deletions, it reconciles subtree move and copy operations, which are not fully supported in other similar work such as [2, 7].

Particularly interesting cases that the `3dm` tool is able to reconcile include combinations of moves and updates, where a node is updated in one tree and moved in another, as well as cases where a subtree was moved in one tree and nodes in that subtree were modified in the other tree. Such cases are examples of what traditional line-based tools cannot handle. Fig. 1 illustrates such a three-way merge case. In T_1 the node v is updated to v' , and the node u is moved. In T_2 a new node a is inserted. `3dm` produces the reconciled tree T_m , which integrates all three operations.

We note that the three-way merge of trees in general is by no means unambiguously defined. In the examples we will use the definition implemented by `3dm`, which was designed to be practically useful and easy to understand.

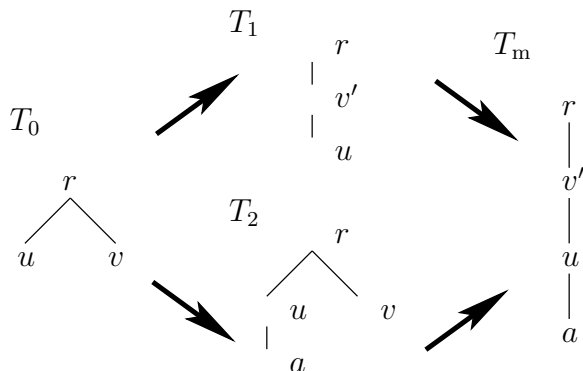


Figure 1: An example of three-way merging of trees. When T_0 , T_1 and T_2 are merged, the result is T_m

The change detection phase of `3dm` is based on *tree matchings* (see e.g. [2]), which identify which node in T_0 a node in T_1 or T_2 corresponds to. These matchings are subsequently used to detect changes. To ensure successful merging we want the change detection to be as accurate as possible, implying that the tree matching should be as accurate as possible. `3dm` includes several tree matchers, e.g. a heuristic $O(n^2)$ matcher for general XML documents, and a fast and accurate $O(n)$ matcher, which can be used when the document elements have unique identifiers.

3. THREE-WAY MERGING IN THE MOBILE ENVIRONMENT

When designing a generic reconciliation facility we need to address some hard theoretical and practical problems: we need to describe and deal with the semantics of the data (a task which quickly becomes very complex, see e.g. [5]), as well as agree on a common way of representing the data.

Despite these issues, a generic approach for reconciliation is desirable, especially in the mobile context: mobile applications need to be small, which argues against adding reconciliation logic to each application individually. Also, if most mobile applications are relatively simple, this is likely to be reflected in the data they process. Hence, semantic issues during reconciliation should be simpler compared to desktop applications.

We claim that reconciliation using XML TMAs has the following advantages:

- relatively small impact on application development
- a flexible way for an application to provide reconciliation capabilities

These advantages should be gained by use of any XML TMA, but to varying degrees depending on the implementation (e.g. some XML TMAs may require unique element identifiers, mandating a certain development effort). In the scope of this paper we are interested in these general properties of XML TMAs, and will therefore not describe *how* a particular TMA merges documents.

3.1 Small Impact on Application Development

A very large class of applications manipulate *files* which store a *state* (as opposed to an edit script), e.g. editors for

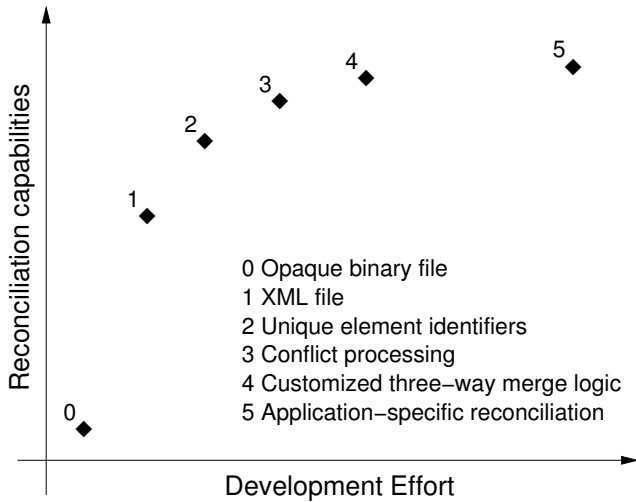


Figure 2: Reconciliation capabilities versus development effort

ASCII text, HTML, XML, or graphics formats such as SVG. We find that reconciliation through three-way merging is a good match for such applications: it is also based on state rather than edit scripts, and the file provides a convenient unit of reconciliation.

A reconciliation engine based on edit scripts would require the application to log the edit operations performed on the data they manipulate. Furthermore, the edit script would need to be encoded in some standardized format in order to be accessed. If not already a part of the application, the maintenance of edit scripts will complicate the design of the application.

A three-way merge approach will affect the application design as well, but generally to a lesser degree: We still need to expose the structure of the data to the reconciliation engine, but we need not log edits.

A pragmatic choice for expressing structured data is XML [1], although there are efficiency issues with mobile devices. These issues are, however, being addressed. Should a more suitable alternative emerge, that may be used just as well.

3.2 Offering Flexibility

Reconciliation based on three-way merging makes it especially easy for application developers to take an incremental approach to supporting reconciliation. With little or no coding effort basic reconciliation capabilities can be provided. By incrementally adding reconciliation capabilities to the application, its ability to function in a mobile environment successively increases.

Fig. 2 sketches the tradeoff between implementation effort and reconciliation support. No effort yields no capabilities for reconciliation, as illustrated by an application storing data in an opaque binary format (point 0 in the diagram). On the other end of the spectrum, we have an application with an application-specific reconciliation engine (point 5). This provides superior support for reconciliation (at least in theory), but at maximum development effort. An optimal reconciliation solution lies in the upper left corner of the diagram: full support with no development effort.

We claim that XML three-way merging positions itself fa-

vorably in this diagram. Firstly, the reconciliation support provided by three-way merging rapidly increases with coding effort. Secondly, a number of different levels of reconciliation support in applications can be implemented, allowing the application developer to choose an appropriate tradeoff between coding effort and reconciliation capabilities. This is illustrated by the following data points in the diagram:

1. The basic step for supporting reconciliation is storing data in an open format (XML). This can usually be implemented with little or no effort, yet it yields considerable reconciliation support.
2. Providing unique identifiers for the elements in the stored XML document vastly increases the accuracy and speed of the tree matching phase. Usually such identifiers can be provided by quite modest coding effort.
3. Conflict handling. The application processes the conflict information emitted from the reconciliation engine.
4. Customized reconciliation logic in combination with the generic reconciliation algorithm. We apply the application’s knowledge of semantics to improve the reconciliation, as illustrated in section 4.1.

We observe that the three-way merging “road map” for better reconciliation capabilities indicates that during initial development of reconciliation support, small increases in application code rapidly yield better capabilities. If we were to implement application-specific reconciliation, the effort to get even rudimentary support would in many cases be considerable, and the highest returns on coding effort would likely occur during the final steps towards implementing full reconciliation support.

Finally, we note that the flexibility of three-way merging may help us to write adaptive applications: if resources are too limited, we can drop the level of support for reconciliation by e.g. not downloading the customized reconciliation logic for an application.

4. EXAMPLES

The use of an XML TMA for data reconciliation is illustrated in the mobile file system being developed in the Fuego Core research project. The file system uses `3dm` as an engine for reconciling XML files when synchronizing with another device.

As an example of the type of small, useful mobile applications, that we envision will benefit from reconciliation by XML three-way merge, we have written a shared photo library applet. The photos in the library are stored as JPEG files in a directory along with an XML index file which contains metadata for each photo: title, keywords, etc. The file system knows that the index file contains XML, and automatically maintains a base version as well as reconciles any concurrent updates made to it using `3dm`. Hence, by simply storing the index file in XML format, we enable automatic reconciliation of concurrent edits to the photo library in many cases.

Although cursory, this example illustrates the central idea: reconciliation capabilities can be gained with very little development effort.

```

<?xml version="1.0">
<tree id="1">
  <directory name="tma" id="01">
    <directory name="fig" id="10">
      <file name="s-closure.tiff" id="20" />
      <file name="simple-ex.tiff" id="21" />
      <file name="photolib.tiff" id="23" />
    </directory>
    <file name="paper.lyx" id="11" />
  </directory>
  <directory name="src" id="22">
    <file name="VersionHistory.java" id="08" />
  </directory>
</tree>

```

```

<?xml version="1.0">
<tree id="1">
  <directory name="tma" id="01">
    <directory name="fig" id="10">
      <file name="s-closure.tiff" id="20" />
      <file name="simple-ex.tiff" id="21" />
    </directory>
    <file name="paper.lyx" id="11" />
  </directory>
  <file name="VersionHistory.java" id="08" />
</tree>

```

```

<?xml version="1.0">
<tree id="1">
  <directory name="tma" id="01">
    <directory name="fig" id="10">
      <!-- s-closure.tiff deleted -->
      <file name="simple-ex.tiff" id="21" />
    </directory>
    <file name="paper.lyx" id="11" />
  </directory>
  <file name="History.java" id="08" />
</tree>

```

Figure 3: XML directory trees. The base tree is in the middle. The trees to the left (on device A) and to the right (on device B) are modifications of the base tree. Changes are marked in *slanted style*.

4.1 Directory Tree Reconciliation

The Fuego Core mobile file system also uses three-way merging of XML internally to reconcile updates to the directory hierarchy. The directory hierarchy of the file system is expressed as an XML document. When the file system is synchronized, a three-way merge of the local and remote directory hierarchies (changed trees) and the unified directory hierarchy (base tree), which was obtained after the last synchronization, is performed.

The main advantage of this approach is that very little specialized directory reconciliation logic needs to be written, as 3dm readily implements all the basic functionality. We automatically gain the ability to reconcile involved combinations of file system updates. For instance, renaming a file on one device and moving it on another presents no problem: upon reconciliation the file is both moved and renamed.

The XML representation of directory trees we use is illustrated in Fig. 3. The structure of the document expresses the directory hierarchy, with <directory> elements for directories and <file> elements for files. Each element has a unique identifier to make the tree matching phase of the merge fast and accurate.

As an example of directory reconciliation, assume that the middle directory tree in Fig. 3 has been modified during disconnection of devices A and B as shown in the figure. On device A the Java source files have been moved into a directory called src, and a new file photolib.tiff has been added into the fig directory. On B, VersionHistory.java has been renamed to History.java and s-closure.tiff has been deleted from the fig directory. By performing a three-way merge of the trees in Fig. 3, we get the reconciled directory tree, as shown in Fig. 4.

3dm does not, however, handle all situations to our satisfaction without some additional application-specific reconciliation logic. The file system adds the following logic to the reconciliation process:

- 3dm currently only handles ordered trees, whereas the directory trees are unordered. Fortunately, this is easily resolved: we simply choose to ignore any ordering conflicts arising from the reordering or insertion of child elements inside a directory, in which case 3dm uses the ordering from either tree—a valid approach in this case.
- Names inside a directory need to be unique. This is im-

```

<?xml version="1.0">
<tree id="1">
  <directory name="tma" id="01">
    <directory name="fig" id="10">
      <file name="simple-ex.tiff" id="21" />
      <file name="photolib.tiff" id="23" />
    </directory>
    <file name="paper.lyx" id="11" />
  </directory>
  <directory name="src" id="22">
    <file name="History.java" id="08" />
  </directory>
</tree>

```

Figure 4: Reconciled directory tree.

plemented as a post-processing phase of the three-way merge, where duplicate names are removed by changing the name attribute of either element.

Reconciliation fails if there are conflicts and warnings emitted by 3dm that are not handled by the directory reconciliation logic. In this case the user must resolve conflicts manually. This happens when e.g. the same directory has been moved to different locations on devices A and B.

This example demonstrates that although 3dm will not by default always produce the correct result, we were able to take significant advantage of it: we did not have to write any code to do the actual reconciliation. Furthermore, by adding some rather simple steps to the reconciliation process, we are able to customize the reconciliation process to work properly on directory trees in all situations. Also, the 3dm conflict vocabulary helped us to identify and take proper actions in different cases of conflicts.

5. RELATED WORK

Work on automatic reconciliation has mainly concentrated on databases and reconciliation of edit scripts. Considerably less research exists on generic approaches to state-based reconciliation of application data stored in files.

The Coda [12] file system detects concurrent modifications of files, but does not include a generic data reconciliation algorithm. However, application-specific reconciliation algorithms can be associated with different file types. To reconcile changes pertaining to the directory structure of files, Coda uses a custom algorithm. Coda represents the

typical approach taken in distributed file systems of leaving the reconciliation task to application-specific modules.

The Concurrent Versions System (CVS) [4] supports two types of files: binary and text. On text files, CVS uses a Unix diff3-like algorithm to reconcile changes. Conflicts must be resolved manually by the user. CVS is perhaps the most well-known system with generic (although limited) reconciliation capabilities.

In [6] Kohlhasse and Anghelache propose an infrastructure for collaboration on mathematical knowledge. The infrastructure consists of extensions to the CVS model, one of which is to add support for structured data in XML format. The authors propose that XML TMAs could be used for reconciliation.

A mobile data synchronization framework implemented in Java is described in [3]. The unit of synchronization in this framework is the Java object. Reconciliation of objects is application-specific, which in this case means that each object needs to implement a Java interface containing the reconciliation logic for the object.

The XMiddle [10] XML-based middleware for peer-to-peer computing provides a framework for synchronization and reconciliation of mobile data. Reconciliation is based on two-way merging of XML documents: An XML differencing tool is used to identify the differences between concurrently modified copies. The set of differences is then used to detect conflicts, optionally process these with application-specific merging logic, and produce the merged version.

Interestingly, there seems to exist relatively little work on three-way merging of general tree-structured data, perhaps due to the lack (until now) of a widely adopted format for expressing such data, the difficulty of designing efficient tree matching algorithms, and the general opinion that successful merging requires knowledge of application-specific semantics.

In [5] Horwitz et al. present a TMA for reconciliation of a restricted class of computer programs. The algorithm well illustrates the complexities of designing a reconciliation algorithm for structured data with full knowledge of the semantics of the data.

An SGML/XML merging algorithm designed for use on technical documentation that is able to integrate an arbitrary number of documents into one is presented in [9]. The operation of the algorithm can, however, not be understood as the integration of *changes* with respect to a base version into a merged version, and thus it positions itself outside the definition of three-way merging used in this paper.

The DeltaXML [7] tool handles both two-way merge and three-way merge of XML. In the three-way merge case it is possible to do automatic reconciliation of deletions, updates, and inserts. DeltaXML does not handle move or copy operations.

6. CONCLUSIONS AND FUTURE WORK

In this paper we argued for the suitability of XML TMAs, as exemplified by the 3dm tool, for reconciliation of data in a mobile environment. For the large class of applications that store state in files, the effect on the way such applications are developed should be small. Furthermore, TMAs offer a *flexible* way for an application to implement reconciliation capabilities.

Two examples of using XML three-way merging as implemented by 3dm for reconciliation were presented: a simple

shared photo library and the directory tree synchronization logic in a mobile file system.

The photo album demonstrated how entry-level reconciliation capabilities were gained just by designing an application to store its data in XML. In the directory tree example, we needed maximal reconciliation capabilities, and therefore had to add some domain-specific logic. Still, we were able to make considerable use of the basic 3dm algorithm.

In the future we plan to investigate the use of declarative reconciliation policies for application data formats. For instance, in the directory tree example it would be beneficial if we could inform the reconciliation engine of the fact that the tree is unordered.

7. REFERENCES

- [1] W3C. *Extensible Markup Language (XML) 1.0*, 2nd edition, October 2000. [Recommendation] <http://www.w3.org/TR/2000/REC-xml-20001006/>.
- [2] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 493–504, 1996.
- [3] N. H. Cohen. A Java framework for mobile data synchronization. In O. Etzion and P. Scheuermann, editors, *Cooperative Information Systems: 7th International Conference (CoopIS 2000)*, pages 287–298, Berlin, September 2000. Springer-Verlag.
- [4] *Concurrent Versions System: The open standard for version control*. <http://www.cvshome.org>.
- [5] S. Horwitz, J. Prins, and T. Reps. Integrating noninterfering versions of programs. *ACM Transactions on Programming Languages and Systems*, 11(3):345–387, 1989.
- [6] M. Kohlhasse and R. Anghelache. Towards collaborative content management and version control for structured mathematical knowledge. In *Proceedings of Second International Conference on Mathematical Knowledge Management (MKM)*, 2003. To appear.
- [7] R. la Fontaine. Merging XML files: a new approach providing intelligent merge of XML data sets. In *Proceedings of XML Europe 2002*, May 2002.
- [8] T. Lindholm. A 3-way merging algorithm for synchronizing ordered trees — the 3DM merging and differencing tool for XML. Master’s thesis, Helsinki University of Technology, Dept. of Computer Science, September 2001. <http://www.cs.hut.fi/~ctl/3dm/thesis.pdf>.
- [9] G. W. Manger. A generic algorithm for merging SGML/XML-instances. In *Proceedings of XML Europe 2001*, Berlin, 2001.
- [10] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Personal and Wireless Communications*, April 2002. Kluwer.
- [11] Y. Saito and M. Shapiro. Replication: Optimistic approaches. Technical Report HPL-2002-33, Hewlett Packard Laboratories, February 2002.
- [12] M. Satyanarayanan and J. Kistler. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.