Peer-to-peer collaboration over XML documents

Claudia-Lavinia Ignat and Gérald Oster

LORIA, INRIA Nancy-Grand Est, Nancy Université, France
{ignatcla,oster}@loria.fr

Abstract. Existing solutions for the collaboration over XML documents are limited to a centralised architecture. In this paper we propose an approach for peer-to-peer collaboration over XML documents where users can work off-line on their document replica and synchronise in an ad-hoc manner with other users. Our algorithm for maintaining consistency over XML documents recursively applies the tombstone operational transformation approach over the document levels.

Keywords: XML, collaborative editing, peer-to-peer collaboration, operational transformation

1 Introduction

Collaboration is a key requirement of teams of individuals working together towards some common goal. Computer-supported collaboration is becoming increasingly common, often compulsory in academia and industry where people work in teams and are distributed across space and time. Extensible Markup Language (XML) is a popular format for marking up various kinds of data, such as application data, metadata, specifications, configurations, templates, web documents and even code. In many engineering fields such as geospatial, architectural, automotive and product engineering users interact with the main artifacts of this field by means of certain specialised tools but the underlying representation of these artifacts is done in XML. Many different companies are involved in the engineering process and all subcontractors and suppliers have the need to collaborate over the same data. Existing solutions for the collaboration over XML documents are limited to a centralised architecture. Locking and turn-taking approaches that allow only one active participant at a time are examples of very basic solutions. Another typical solution for editing XML documents is the asynchronous collaboration where users work in isolation on their copies of the document and synchronise their changes against a shared repository where changes are published.

In this paper we propose an approach for the reconciliation of XML documents in a decentralised peer-to-peer (P2P) environment where users can work in an off-line mode and synchronise in an ad-hoc manner with other members of the network without publishing their changes on a central repository. Off-line work means that users can work disconnected from the network, for instance

on their laptops while traveling on train or plane and synchronise their changes when they reconnect to the network. Ad-hoc collaboration would allow members of a team working on a common project to synchronise and revise their changes without making them public to the other teams involved in the project.

For supporting off-line work all or part of the content has to be replicated on multiple peers. Peers can concurrently modify their replicas. A main issue is how to maintain consistency in the face of concurrent modifications on the replicated XML content.

As opposed to centralised version control systems such as Concurrent Versions System (CVS) and Subversion, in P2P environments merging of concurrent changes is not performed by humans. A merging performed by humans in a P2P environment might lead to a same situation of conflict resolved in different ways by different users. In order to ensure convergence in a P2P environment merging of concurrent changes is automatically performed. Users can revise the result of merging to correct the possible conflictual changes by generating new operations.

In this paper we propose an automatic algorithm for merging XML documents by combining the treeOPT [4] approach for merging hierarchical structures with the tombstone transformational approach [7] modified for XML documents. The XML document obtained as result of merging should be well-formed. To our knowledge, our approach is the first one that can reconcile XML documents in a P2P environment.

The paper is structured as follows. In section 2 we present existing approaches for the reconciliation of XML documents. Section 3 describes the model of the document and the set of operations for modeling user modifications on the document. We then go on in section 4 by presenting the treeOPT algorithm used in our approach for the reconciliation of hierarchical structures. Section 5 presents the transformation functions for XML documents used by treeOPT algorithm recursively over the document hierarchy. In section 6 we present our concluding remarks and directions of future work.

2 Related work

Various approaches were proposed for reconciliation of XML documents. They can be classified into state-based and operation-based approaches.

State-based approaches use only information about document states and no information about the evolution of one state into another. Generally, these approaches rely on a diff algorithm [2] for computing the changes performed between two document states. These diff algorithms are not reliable as there is usually more than one function that can be used to transform an initial state of document into a final one. Moreover, they do not keep information about the process of transformation from one state to the other, such as execution order of operations. The algorithms for the computation of XML difference are costly in terms of resources. Merging used by these approaches relies on a document reference copy and therefore it is not commutative and associative, the necessary conditions for synchronisation in decentralised environments. Subversion, CVS

and distributed version control systems use state-based approaches for document merging.

As opposed to state-based approaches, operation-based approaches [6] keep information about the evolution of one document state into another in a buffer containing the operations performed between the two document states. This approach allows tracking accurate user changes and capturing operations' semantic. On the other side, the used editing tool should be able to capture the performed operations. Our approach belongs to the category of operation-based approaches.

The operational transformation [3] approach has been identified as an appropriate approach for maintaining consistency of the copies of the shared document in operation-based collaborative editing systems. It allows local operations to be executed immediately after their generation and remote operations need to be transformed against the other operations. Transformations are performed in such a manner that user intentions are preserved and, at the end, document copies converge.

Currently, it is possible to synchronise data in a decentralised environment, but data has to conform to a linear structure. For instance, in the tombstone transformational approach [7] a text document is seen as a sequence of characters or lines. In [10] an approach for maintaining consistency over CoWord and CoPowerPoint documents was proposed. Documents conform to a hierarchical structure, but update operations are simulated as a delete followed by an insert with the new value. As mentioned in section 5, this solution is not satisfying for our application as it would lead to not well-formed documents. Furthermore, the approach proposed in [10] was not applied for XML documents.

Approaches for synchronisation of XML documents exist, but they are limited to a centralised architecture. Some of the approaches that use a central server for the synchronisation of XML documents can be found in [8, 5].

3 Document model and operations definition

This section presents our document model and the operations for describing user changes on the document.

A node N of a document is a structure of the form $N =< parent, children, attributes, history, content >$, where

- *parent* is the parent node for the current node
- *children* is an ordered list $[child_1, ..., child_n]$ of child nodes
- *attributes* is a set of attributes, each attribute being defined as a pair (*name*, *value*), where *name* is the name of the attribute and *value* is its associated value
- *history* is an ordered list of operations executed on child nodes
- *content* is the textual content of the node if this node is a textual node. Otherwise, it represents the name of the node.

The level of a node is its depth in the tree, i.e. the length of the path from root to the node.

The child nodes of a node N are ordered and therefore they will be relatively identified to node N by their position in its list of children. The absolute position of a node N is the path defined by the relative positions of all its ancestors.

There is no order assigned for the attributes of a node in an XML document, and therefore, attributes of a node N in our model are unordered. Moreover, a well-formed XML document requires that an element does not have two attributes with the same name. Therefore, we have chosen to identify an attribute of a node N by its name in the set of attributes of node N.

In order to describe user changes performed on the structure of the document, we defined the following operations:

- *insertNode(p,content)* that inserts a node at the absolute position denoted by path p with the content *content* (the meaning of a node content was previously defined).
- *deleteNode(p)* that deletes the node identified by the path position p.
- *setAttribute(p,name,value)* that assigns the value *value* to the attribute identified by the name *name* of the node with path p. If the attribute with name *name* does not exist, it is created.

Note that we simulate a deletion of an attribute with name *name* belonging to the node with path p as $setAttribute(p, name, null)$. When such an operation is executed on an XML document, the corresponding attribute is removed.

4 The treeOPT algorithm

This section presents the treeOPT algorithm that we used for maintaining consistency over the XML hierarchical structure.

Each user peer maintains a local copy of the hierarchical structure of the XML document. Local operations generated by a peer are immediately executed and added to the local history buffers distributed throughout the tree. In a synchronisation phase, remote operations generated by other peers are transmitted to the local peer. For the integration of a remote operation into the corresponding history buffer, the treeOPT approach recursively applies over the hierarchical document an existing operational transformation algorithm for linear structures, such as SOCT2 [9]. In what follows we briefly present the implementation of the treeOPT algorithm using the SOCT2 algorithm. A more detailed explanation of the treeOPT algorithm can be found in [4]. We call a composite operation an operation on the tree whose position is defined by an absolute position in the tree. For each relative position in the path, the composite operation has a corresponding simple operation defined on one level of the tree.

Algorithm *treeOPT-SOCT2(O, RN, L)* {
  $CN = RN$;
  for ($l := 1$; $l \leq L$; l++) {
    $O_{new} := Composite2Simple(O, l)$;
    $EO_{new} := SOCT2(O_{new}, history(CN))$;
    $position(O)[l] := position(EO_{new})$;

```
        if (level(O) = l) {
            Do O;
            Append(EO_new, history(CN));
        }
        CN = child_i(CN), where i = position(EO_new);
    }
}
```

Given a new causally ready composite operation, O, the root node of the hierarchical representation of the local copy of the document, RN, and the number of levels in the hierarchical structure of the document, L, the execution form of O is computed. Determining the execution form of a composite operation requires finding the elements of the position vector corresponding to a coarser or equal granularity level than that of the composite operation. For each level of granularity l, starting with root level and ending with the level of the composite operation, the SOCT2 linear merging algorithm is applied to find the execution form of the corresponding simple operation. SOCT2 does not perform transformations on composite operations, but rather on simple ones. Therefore, we had to define the function *Composite2Simple*, that takes as arguments a composite operation, together with the granularity level at which we are currently transforming the operation, and returns the corresponding simple operation. The operational transformation algorithm is applied to the history of the current node CN whose granularity level is $l-1$. The lth element in the position vector will be equal to the position of the execution form of the simple operation. If the current granularity level l is equal to the level of the composite operation O, O is executed and its simple operation corresponding to level l is appended to the history of the current node CN. Otherwise, the processing continues with the next finer granularity level, with CN being updated accordingly. The function $SOCT2(O, HB)$ takes as parameters a causally-ready simple operation O and a history buffer HB and returns the execution form of O.

5 Tombstone transformation functions

As we have seen in the previous section, treeOPT approach recursively applies an operational transformation algorithm working for linear structures over the hierarchical document levels. Our method was to combine treeOPT with a linear operational transformation algorithm that works in peer-to-peer networks. As far as we are aware, the tombstone transformation functions [7] are the only ones that satisfy the mandatory consistency properties [9]. Satisfying these conditions ensures that the merging algorithm is associative and commutative, i.e. synchronisation between a set of peers can be performed in any order. Therefore, the tombstone transformational approach can be safely used for merging documents conforming to linear structures (e.g. a text document is seen as a sequence of characters) in a distributed environment.

We therefore chose to combine the tombstone transformational approach with the treeOPT approach and in this way treeOPT-SOCT2 is the first operational

transformation algorithm applicable in peer-to-peer environments that maintains consistency over replicated hierarchical documents.

We next present the novel transformation functions for XML documents that we designed for the transformations to be applied at the level of a certain node N. These operations are either *insertNode* or *deleteNode* operations that insert or respectively delete child nodes of node N or *setAttribute* that sets the value of a certain attribute of node N. Additionally, each operation contains a parameter *sid* representing its generator site. Transformation functions are written for each pair of operations.

T $(insertNode(p_1, content_1, sid_1), insertNode(p_2, content_2, sid_2))$
if $(p_1 < p_2)$ return $insertNode(p_1, content_1, sid_1)$
else if $(p_1 = p_2$ and $sid_1 < sid_2)$ return $insertNode(p_1, content_1, sid_1)$
else return $insertNode(p_1 + 1, content_1, sid_1)$

T $(insertNode(p_1, content_1, sid_1), deleteNode(p_2, sid_2))$
return $insertNode(p_1, content_1, sid_1)$

T $(insertNode(p_1, content_1, sid_1), setAttribute(p_2, name_2, value_2, sid_2))$
return $insertNode(p_1, content_1, sid_1)$

T $(deleteNode(p_1, sid_1), insertNode(p_2, content_2, sid_2))$
if $(p_1 < p_2)$ return $deleteNode(p_1, sid_1)$
else return $deleteNode(p_1 + 1, sid_1)$

T $(deleteNode(p_1, sid_1), deleteNode(p_2, sid_2))$
return $deleteNode(p_1, sid_1)$

T $(deleteNode(p_1, sid_1), setAttribute(p_2, name_2, value_2, sid_2))$
return $deleteNode(p_1, sid_1)$

T $(setAttribute(p_1, name_1, value_1, sid_1), insertNode(p_2, content_2, sid_2))$
if $(p_1 < p_2)$ return $setAttribute(p_1, name_1, value_1, sid_1)$
else return $setAttribute(p_1 + 1, name_1, value_1, sid_1)$

T $(setAttribute(p_1, name_1, value_1, sid_1), deleteNode(p_2, sid_2))$
return $setAttribute(p_1, name_1, value_1, sid_1)$

T $(setAttribute(p_1, name_1, value_1, sid_1), setAttribute(p_2, name_2, value_2, sid_2))$
if $(name_1 = name_2)$
    if $(site_1 < site_2)$ return $setAttribute(p_1, name_1, value_1, sid_1)$
    else return $setAttribute(p_1, name_1, value_2, sid_1)$
else return $setAttribute(p_1, name_1, value_1, sid_1)$

An *insertNode* operation transformed against another *insertNode* operation keeps its original form if the insertion position of the second *insertNode* operation is situated after the insertion position of the first operation or the two operations have the same insertion position and the generation site of the first operation is smaller than the generation site of the second operation. Otherwise the transformed operation increases by 1 its insertion position. Note that according to SOCT2 algorithm only concurrent operations are transformed against each other. Therefore, two operations generated by the same site are never transformed against each other.

A *setAttribute* operation transformed against a *setAttribute* operation keeps its original form if the targeted attributes are different or they are the same but the generator site identifier of the first operation is smaller than the generator

site identifier of the second operation. Otherwise, the attribute value becomes equal to the value set by the second *setAttribute* operation.

Any operation transformed with respect to a delete operation *deleteNode* keeps its original form as deletions of nodes are not physically performed. An *insertNode* and *deleteNode* operations transformed with respect to a *setAttribute* operation keep their original form since modifications of attributes do not change the hierarchical structure of XML elements.

A *deleteNode* and *setAttribute* transformed with respect to an *insertNode* operation keep their original form if the insertion position of *insertNode* is situated after the position targeted by *deleteNode* or *setAttribute*. Otherwise, their position is increased by 1.

The need to resolve conflict between two concurrent *update* operations referring to the same entity is a major challenge for the operational transformation approach. For a linear structure, transformations among *insert* and *delete* operations simply adjust the positions of these operations. There is no conflict between these two types of operations, their effect being always preserved by adapting their positional parameter. By using tombstone transformational approach, the effect of *insert* and *delete* operations can be preserved also in the case of a hierarchical structure. The tombstone transformational approach does not remove entities from the document structure, just marks them as invisible. This fact is helpful in the process of automatic merging of concurrent changes for highlighting modifications concurrently performed. If nodes are not removed from the document structure, we can highlight concurrent changes of deletion of a node and of a change performed on the deleted subtree. However, the transformation technique for *insert* and *delete* operations is no longer relevant to the transformation of *update* operations. In our case, the *update* operation is the *setAttribute* operation. Two update operations targeting the same entity and attribute are considered conflicting as there is no way to satisfy the effect of both operations without changing the structure of the document. A solution to this situation of conflict is to create versions of the targeted attribute, but then the issue is how to present versions to the user. Another solution, the one that we adopted, is to ignore the effect of one of the two conflicting updates by setting its value to the value of the other update. Another alternative solution is to simulate the *update* operation as a deletion followed by an insertion of attribute. In our case, this solution would lead to a not well-formed XML document with an element having two attributes with the same name and different values.

6 Conclusion

In this paper we proposed an approach for peer-to-peer collaboration over XML documents. Users can work off-line on their document replicas and synchronise in an ad-hoc manner with other users. Our approach combines the treeOPT approach for merging hierarchical structures with the tombstone transformational approach that we adapted for XML documents. Our approach recursively applies the tombstone operational transformation approach for linear structures

over document levels. To our knowledge, our approach is the first one that can reconcile XML documents in a P2P environment.

Our proposed approach relies on the notion of state vectors for detecting concurrency between operations. State vectors impose the limitation of closed groups and cannot be used for supporting dynamic massive collaborative editing involving a large number of users who can often join and leave the network. In order to adapt our approach to support massive collaboration we will investigate how to combine treeOPT with MOT2 [1] approach. MOT2 approach is based on operational transformation, but instead of state vectors it uses a pair-wise synchronization mechanism according to which it constructs a common history of operations for all sites.

References

1. M. Cart and J. Ferrié. Asynchronous reconciliation based on operational transformation for P2P collaborative environments. In *Proceedings of the International Conference on Collaborative Computing (CollaborateCom'07)*, White Plains, New York, USA, November 2007.
2. G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, pages 41–52, San Jose, California, USA, February 2002.
3. C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. *SIGMOD Record*, 18(2):399–407, June 1989.
4. C.-L. Ignat and M. C. Norrie. Customizable collaborative editor relying on treeOPT algorithm. In *Proceedings of the 8th European Conference on Computer-supported Cooperative Work (ECSCW'03)*, pages 315–334, Helsinki, Finland, September 2003.
5. C.-L. Ignat and M. C. Norrie. Flexible Collaboration over XML Documents. In *Proceedings of the International Conference on Cooperative Design, Visualization and Engineering (CDVE'06)*, pages 267–274, Mallorca, Spain, September 2006.
6. E. Lippe and N. van Oosterom. Operation-based merging. In *Proceedings of the ACM SIGSOFT Symposium on Software Development Environments (SDE'92)*, pages 78–87, Tyson's Corner, Virginia, USA, December 1992.
7. G. Oster, P. Molli, P. Urso, and A. Imine. Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems. In *Proceedings of the International Conference on Collaborative Computing (CollaborateCom'06)*, page 10, Atlanta, Georgia, USA, November 2006. IEEE Computer Society.
8. G. Oster, H. Skaf-Molli, P. Molli, and H. Naja-Jazzar. Supporting Collaborative Writing of XML Documents. In *Proceedings of the International Conference on Enterprise Information Systems (ICEIS'07)*, pages 335–342, Funchal, Madeira, Portugal, June 2007.
9. M. Suleiman, M. Cart, and J. Ferrié. Serialization of concurrent operations in a distributed collaborative environment. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work (GROUP'97)*, pages 435–445, Phoenix, Arizona, USA, November 1997. ACM Press.
10. C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai. Transparent Adaptation of Single-user Applications for Multi-user Real-time Collaboration. *ACM Transactions on Computer-Human Interaction*, 13(4):531–582, 2006.