Inconsistency of XML Documents during Cooperative Editing

Hala Skaf-Molli, Hala Naja-Jazzar, Pascal Molli

▶ **To cite this version:**

Hala Skaf-Molli, Hala Naja-Jazzar, Pascal Molli. Inconsistency of XML Documents during Cooperative Editing. [Research Report] 2006. inria-00123346

HAL Id: inria-00123346
https://inria.hal.science/inria-00123346

Submitted on 9 Jan 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Inconsistency of XML Documents during Cooperative Editing

Hala Skaf-Molli (1), Hala Naja-Jazzar (2), and Pascal Molli(1)

[1] LORIA- INRIA LORRAINE, BP 239, 54506 54506 Vandœure-Les-Nancy, France
[2] University of Liban skaf@loria.fr, hjazzar@ul.edu.lb, molli@loria.fr,
WWW home page: http://www.loria.fr/ skaf
[3] http://www.loria.fr/ molli

Abstract. XML-based file format must be validated against its DTD in order to be visualised. In cooperative editing, the replication of XML documents in different sites allows to ameliorate the availability of data. After the reconciliation of the different replicas of the initial document, it is possible that the result of the merging does not validate the DTD. This means that it is not possible to open the document with the tool that edits it. To overcome this problem, we propose an automatic repairing approach in order to re-establish the consistency of XML documents.

1 Introduction

Generally, users involved in collaborative editing work on shared data. In order to achieve high responsiveness, shared data are replicated on all sites. In order to achieve unconstrained interactions, there are no locking or serialization protocols. Any user can edit the document at any time. If two users generate concurrent operations, the system has to ensure that replicas will converge while preserving effects of concurrent operations.

We have developped a framework LibreSource (www.libresource.org). This framework allows asynchronous collaborative editing of text files and XML files. The same algorithm is used to handle text data and XML data. This framework is based on the Operational Transformation (OT) approach and ensures eventual consistency. It means that when the system is idle, all copies are identical.

In this paper, we will illustrate inconsistency problem resulting form synchronising XML data.

XML (eXtensible Markup Language) has become the prime standard for represent, exchange and storage of data. XML document usually comes with a Document Type Definition DTD that specifies the structure of the document. In some way, this is the grammar associated with the document. An XML document is valid or consistent if it conforms to its DTD.

In this paper, we propose a framework that

XML-based file format is more and more used. Many applications use it to store data. This enhance the interoperability with other applications. Traditionaly, XML-based file format environment has a built-in function to verify the validation of the

XML document. If it is not the case an error message is displayed and the user has to modify his/her document else s/he cannot save it.

The validation of the XML-based file format becomes more complex if the document is resulting from the merging of different replicas of a shared document. When several persons working together to edit an OpenOffice document. The shared document is usually replicated in the site of each participant. This allows more availability of the document, in addition, each participant in a cooperative editing has his/her own replica of the OpenOffice file, s/he can work insulated in his workspace. This phase of cooperation produces copies divergence. From times to times copies have to be resynchronised in order to integrate the work of the different participants. During the synchronisation phase, it is possible to produce a document that do not validate the DTD , in spite of the different replicas do it. To illustrate this idea, let us consider the following XML-based file format and its associated DTD, called D.

2 Problem Statement

*Example 1 (*DTD D*).*

```
<!ELEMENT bib (book+)>
<!ELEMENT book (title, auteur+, publisher>
<!ELEMENT auteur (fname?, lname)>
```

D describes the following structural constraints:

1. A bibliography has at least one book.
2. A book has one title followed by at least one author and one publisher.
3. An author has zero or several first name followed by one last name.

Here we omit the descriptions of elements whose type is string (i.e. PCDATA in XML).

*Example 2 (*XML document*).*

```
<bib>
 <book>
  <title>Database</title>
  <author>Ullman</author>
  <author>Widom</author>
  <pub>Prentice</pub>
 </book>
</bib>
```

*Example 3 (*XML document that conforms to D*).*

```
<bib>
 <book>
  <title>  Database Systems </title>
  <auteur>
```

```
    <fname> jeffry </fname>
    <lname> Ullman </lname>
    <fname>  Jennifer  </fname>
    <lname> Widom  </lname>
  </auteur>
  <publisher> Prentice  </publisher>
 </book>
</bib>
```

The XML document of the example 2corresponds to the XML tree of the figure 1.

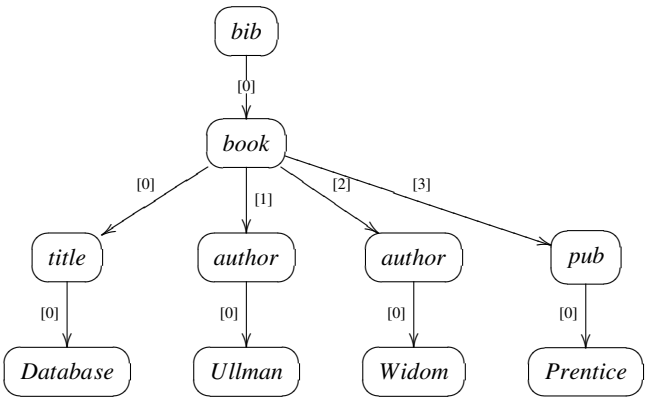

Fig. 1. The XML Tree corresponding to the document of Example 2

In this paper, we suppose that the tree is ordered i.e. the children of every node are ordered, that is, there is a first child, a second child, a third child, etc. Therefore, each node is uniquely identified by its path. On a XML node, we define the following operations:

- addNode(parent, n, val) adds a new node as a children of the node identified by the path *parent*. This node is added as *n*th child and its value (or label) is *val*.
- delNode(parent, n) deletes the *n*th children of the node identified by the path *parent*.

Now, imagine two persons *me* and *you* working together to edit the shared XML-file format. Each has a replica of the shared document. We use the synchroniser So6 to integrate the different modifications. So6 is an XML synchroniser that preserves user intention i.e. does not destroy work done by users and ensures data convergence i.e. after synchronisation all the replicas have an identical value. After synchrnosation, the replica of *me* and *you* have the value mentionned at the third couloun..

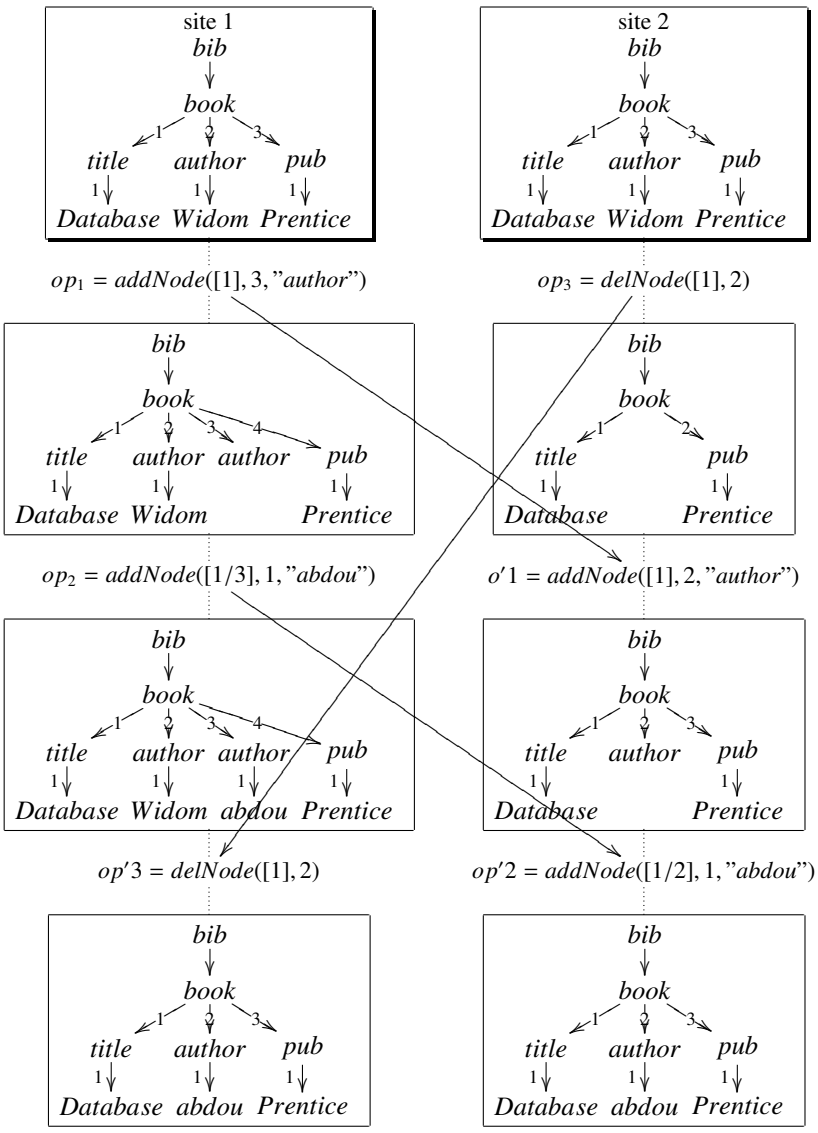

Fig. 2. Integration scenario

Table 1. Local XML Replicas

Me replica	You replicas
<bib>	<bib>
<book>	<book>
<title> Database Systems </title>	<title> Database Systems </title>
<auteur>	<auteur>
<fname> jeffry </fname>	<fname> jeffry </fname>
<lname> Ullman </lname>	<lname> Ullman </lname>
<fname> jen </fname>	<fname> jen </fname>
<lname> Widom </lname>	<lname> Widom </lname>
</auteur>	</auteur>
<publisher> Prentice </publisher>	<publisher> Prentice </publisher>
</book>	</book>
</bib>	</bib>

3 Example

The scenario presented in the figure 2 illustrates how the So6 framework works. In this scenario, there are two users working in two different sites $site_1$ and sit_2. They share the same initial state of the XML document. Each one has a copy of the shared document in his workspace. Users work concurrently to edit the document. The first one produces operations op_1 and op_2 while the second produces the operation op_3. The states of the documents corresponding to these modifications are shown in the figure 2. After that, the second user *commits* his modifications during this step, the operations op_3 is sent to the timestamper. Then, the first user *updates* his workspace in order to integrate modifications done by the second user. During the *update* step, the transformation functions op'_3, op'_1, op'_2 are calculated. At this step, only the operation op'_3 is locally executed. Next, the first user *commits* his modifications. During this step, op'_1 and op'_2 are sent to the timestamper. When the second user calls *update* operation. op'_1 and op'_2 are executed in the $site_2$. At the end of the execution the two copies of the document converge towards a unique value.

The scenario can be summarized as:

$site_1$	$site_2$
op_1	op_3
op_2	
	commit (send op_3 to timestamper)
update (calculate op'_3, op'_1, op'_2)	
commit (send p'_1, op'_2 to timestamper)	
	update (execute op'_1, op'_2)

*Example 4 (*Synchronised XML File*).*

```
<bib>
 <book>
  <title>  Database Systems </title>
```

```
  <auteur>
    <fname>  jeffry </fname>
    <lname> Ullman </lname>
    <fname>  jen  </fname>
    <lname> Widom  </lname>
  </auteur>
  <publisher> Prentice  </publisher>
 </book>
</bib>
```

The above example show that, generally, the validation of DTD cannot be guaranteed after synchrnoization, even if the the synchronised replicas are consistent when considered separately. This means that if the document was modified by using OpenOffice editor, this editor will not be able to open the merged file. In other word, it is not possible to open the document with the tool that used to produce it. Thus, in the presence of an XML document which must satisfy a DTD the issue of managing possible inconsistencies is relevant.

This problem occurs because XML synchronisers such as deltaXML(www.deltaxml.com) and So6 (courgette.loria.fr/projects/so6) [**?**] ensure the convergence of replicas towards an unique content but they do not ensure the consistent of the merged file with respect to the DTD . One possible solution for this problem is to integrate the validation of the DTD in the synchronizer itself. This means that an XML synchronizer must ensure both data convergence and data validation. This approach is used by IceCube[podc02] framework. The main inconvenient of this approach is that on the one hand, IceCube does not respect users intentions, therefore, in order to ensure data validation and convergence some users operations will be deleted.On the other hand, this obligates users to take early decision. For example, if we consider our example, IceCube will obligate the user during the synchronisation to choice one of the title, which may be not easy to make at this point and maybe the user want to keep both titles.

Another possible solution is to allow the convergence towards a state that does not respect necessarly the DTD and to propose repair actions in ordre to re-establish consistency. The issue of managing inconsistency has been exhaustively investigated in databases domainand several techniques based on the computation of minimal sets of insert/delete operations has been proposed [**?**,**?**]. However, these techniques cannot be easily extended to XML document, due to their complex structure and the different nature of constraints in XML context, meangingful DTDs are defined on the structure of documents, and do not have an equivalent counterpart in relational model [**?**]. The document of the previous example can be repaired by performaing the following minimal sets of update operations:

- insert
- delete

Generally, it is possible to repair a document in many distinct ways, thus generating several repaired documents. In repairing documents we prefer to apply repairing actions that, on one the hand, respect the intention of the user and on the other hand, form min-

imal set of changes to the original document. For instance, for inconsistent document of Example 1 ...

In more detail, the proposed actions must satisfy the following requirements:

1. They must respect the intention of the user. This means that the work done by a user can not disappear after repairing inconsistency. A person who participates in collaborative editing will not agree to lost his work because of ”inconsistency” problem.
2. Correct: the application of a repairing action must reduce the number of inconsistency.
3. Efficient: repairing of an inconsistency does not introduce a new one.
4. and finally complete: this means all possible repairing actions that respect the first requirement are produced.

In this paper, we propose an automatic repairing approach in order to establish validation of merged XML document. In order to define the most adequate repairing actions, it is important to identify exactely the cause of violations. The rest of the paper is organised as follows. In section 2, we give a short description of the synchroniser So6 and its synchronisation rules. In section 3, we formalize the definition of DTD and XML document. In section 4, we identify the cause of possible violation. In section 5, we propose new repairing actions. In section 6, we present some related works. Finally, we conclude the paper with a discussion of futur works.

4 The XML synchroniser So6

4.1 So6 Synchronisation Rules

5 XML Trees and DTDs

Many formal definition for XML and DTD [**?**,**?**] have been proposed. We will use the definition given in [**?**].

Definition 1. *DTD is a tuple D = (E, A, P, R, r), where:*

- *E is a finite set of element types;*
- *A is a finite set of attributes, disjoint from E;*
- *for each t ∈ E, R(t) is a set of attributes in A;*
- *r ∈ E and is called the element type of the root.;*
- *for each t ∈ E, P(t) is a regular expression a, called the element type definition of t is defined by:*

$$a ::= S\,|tàu|ea|a?|a|a|a, a|a*|a+$$

where S denotes the string type, *tàu*, e is the empty word, and *tàu·* The sequence, denoted a, b which means that ”a is followed by b”. The alternative, denoted *a|b* which means that ”a or b”. Each element can be mandatory or optional, repetitive or not. We can distinguish the following cases:

Table 2. So6 Synchronisation Rules

	Initial File	Replica 1	Replica2	Synchronized File
Addition in 1	(no element)	<E1 />	(no element)	<E1 />
Addition in 2	(no element)	(no element)	<E1 />	<E1 />
Addition in both, same	(no element)	<E1> 3 </E1>	<E1> 3 </E1>	<E1> 3 </E1> <E1> 3 </E1>
Addition in both,diff	(no element)	<E1> 1 </E1>	<E1> 2 </E1>	<E1> 1 </E1>
				<E1> 2 </E1>
Deletion in 1	<E1>	(no element)	<E1>	<E1><E1>
Deletion in 2	<E1/>	<E1/>	(no element)	<E1/>
Deletion in both	<E1\>	(no element)	(no element)	<E1/>
Contents Change in 1	<E1> 3 </E1>	<E1> 1</E1>	<E1> 3 </E1>	<E1> 3 </E1> <E1> 1 </E1>
Contents Change in 2	<E1> 3 </E1>	<E1> 3</E1>	<E1> 2 </E1>	<E1> 3 </E1> <E1> 2 </E1>
Contents Change in both, same	<E1> 3 </E1>	<E1> R</E1>	<E1> 4</E1>	<E1> 4 </E1> <E1> 4 </E1>
Contents Change in both, different	<E1> 3 </E1>	<E1> 1</E1>	<E1> 2</E1>	<E1> 1 </E1> <E1> 2 </E1>
Contents Change in 1, deletion in 2	<E1> 3 </E1>	<E1> 1 </E1>	(no element)	<E1> 1 </E1>
Contents Change in 2, deletion in 1	<E1> 3 </E1> (no element)	<E1> 2 </E1>		

1. a : the element a is mandatory with only one occurrence.
2. a ? : the element is optional, the cardinality is [0,1]
3. a* : the element is optional and repetitive, the cardinality is [0,N]
4. a+ : the element is mandatory and repetitive, the cardinality is [1,N].

*Example 5 (*Example 1 Revisited*).*

Let us consider the DTD D given in Exampl 1. In our formalism, D can be represented as (E, A, P, R, r) where E=(bib, book, title, autheur, fnam, lname,publisher), A = , r=articles and P is:

P(bib) = book+ P(book) = title, autheur+ , publisher P(autheur) = fname ?, lname P(fname)=S ; P(lname)= S ; P(title) = S.

In this paper, for simplification, we use a restrict form of DTD, only elements of DTD are considered. However, the proposed solution can be apply easly for attributes.

Definition 2. *An XML document is typically modelled as node-labelled tree. In this paper, we suppose that the tree is ordred.*

6 Automatic Reparation Approach

The first step in repairing is to identify the exacte cause of violations.

6.1 Identification of violation causes

It is possible to identify the cause of violation in two ways. The first one is based on analysing of the formal definition of the DTD, especially the regular expression of the elements. The second one is based on the analysing the result of the synchronisation.

Possible Causes Based on DTD We can summerize the following possible violations based on the analysing of the definition of the elements of a DTD:

Table 3. Possible Violation by Grammer Analysing

case	Normal Situation	Possible Violations	Means that
general	a	element a instead of x	
sequence	a,b		
alternative	a—b	nor a, nor b	
a		a does not exists or serveral a	
a?		more than once	
a*			
a+		a does not exist	p(t) = ϵ

The above possible violation can be summarized as:

1. Unicity Violation U: several elements instead of exactley only one. This is the case of a mandatory element of the case 1 and the optional element of case2. Example, a book with several title.
2. Existence Violation E: a mandatory element which does not exist. The elements can be repetitive or not (case 1 or 4). For example, a book without title.
3. Absence Violation A: an element which must not exist. This is the case of an alternative or a new element added after the merging operation. However, the last violation can not occur since this means that individual editing does not validate the DTD.

Analysing the grammar allow to identify all general violation. However, not all of the previous violatioin can occur. For example, absence violation.

A better solution to identify the causes of violation is based on the analysing of the synchronisation rules. This gives better diagnosis of the violation.

Causes Based on Synchronisation Rules The analysing of syncronisation rules allows to identify exactley possible violations and eliminate those cannot occur.

6.2 Repairing Actions

After violation identification, we have to propose repairing action. As we mentioned before an XML document is typically modelled as an ordred node-labelled tree. To repair, we can use the basic tree operations:Add a new node, modify the label of a given node and delete a given node. In some way, these repairing actions are similair of those of databases(insert, delete, update). In the case of an XML document, we can imagine new repairing actions such as:

- Put a node as a commentary ;
- merge one or several nodes totheger;
- move a node to another place ??

Repairing Unicity Violation The traditional solution for this type of violation consists in removing the redundant nodes and keeping only one node. The question is which one ? One can define a system based on priority. The priority can be the name of the person, the date of last modification, etc.. However, such a repair does not respect the intention of the user. A better solution could be:

- merge the different elements to form only one ; or
- keep one element and but the others as a commentary ; or
- keep only one element and put the others as subelements. For example, in the case of several titles we can keep one and put the others as sub-titles. If we apply this solution to the DTD of the example 1, we have to modify the original DTD. We do not study this solution for instant but it can be a future work.

Repairing Existence Violation The only possible solution is to add a new element with a default value.

6.3 Repairing Absence Violation

As we said before, this violation can not occur in our case, since we suppose that individual modfications do not introduce inconsistency.

7 Related Work

A lot of work has been done in XML domain in order to validate [**?**] and merge XML documents []. Non of existing tools integrate both aspects. For example, deltaXML (www.deltaxml.com) propose a set of tools to diff and merge XML documents. Diff produces the difference between two XML documents and merge allows to apply this difference on the original XML document in order to produce a new one. We have to look to the technical FAQ (www.deltaxml.com/core/deltaxml-technical-faq.html) to see that if the document has a DTD so the result of the merge does not eventually validate the DTD. Most of existing tools and validating algorithms allow to detect the violation of the DTD but they do not provide any repairing action. Of course, it will be interesting to use these algorithms for detecting violation.

Xlinkit [**?**,**?**] is a framework to manage consistency of distributed XML document. It defines its own constraints languages. This language is based on first order logic. Xlinkit proposes an algorithm to verify constraints. The result of this algorithm is a set of hyperlinks, in the form of linkbase,. These allow to navigate among inconsistent elements of an XML document. Xlinkit propose also a repairing algorithm which exploit the results provided by the verification algorithm. The approach proposed by Xlinkit is very interesting, however, it proposes only primitive repairing actions (insert, delete update) and do not preserve user intentions.

Recently, Flesca. et al. [**?**] proposes to repair inconsitency of XML document in presence of DTD and integrity constraints. They define three repairing strategies: general repairs, cleaning repairs and completing repairs. The first strategy propose to use a set of insert and delete operation to repair. The second one propose to remove pieces of information in order to clean dirty data. And the last one propose to add data to original document in order to repair inconsistency. As in previous mentionned work, their repairing actions do not satisfy users intentions.

8 Conclusion

We propose an automatic repairing approach to ensure consistency of merged XML document. In case of the violation of the DTD a set of repairing actions are proposed. The proposed actions satisfy the intention of the users. A lot of techniques has been proposed in the domain of database. However, these techniques cannot easily extened to XML document.None

We continue to work on our proposition. We need to formalise our approach and verify that proposed repairing actions satisfy the requirements described in this paper. After the maturity of the proposition we want to experiment it and to adapted to XML schema (or RELAX/NG schemas). We plan to implement and test our algorithm in the framework libresource. In the future, we want to exploit another possible direction to

repair violation. The idea is to extract a new DTD from the merged XML document as in [**?**] and to adapt this new DTD to user intention.

In this work, we are interested in validation of XML document exclusively against its DTD rather than schema for several raisons : DTD is easier to define and understand than schema [workshopdb05]. The validation against DTD is built-in in most of XML compiler. DTD is widely used. For example the administration of the state of Massachusetts The state of Massachusetts plans to move to an open format for office documents.

Notes and Comments.