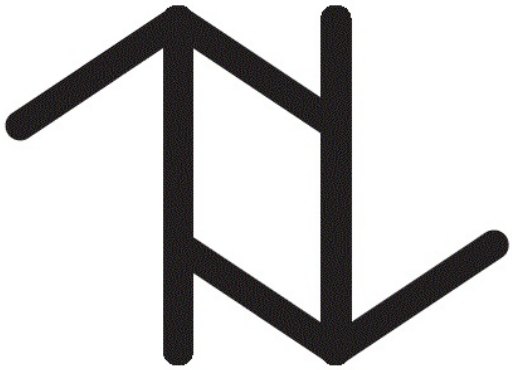

An Efficient Approach for Detecting and Repairing Data Inconsistencies Resulting from Retroactive Updates in Multi-temporal and Multi-version XML Databases

Hind Hamrouni, Zouhaier Brahmia, Rafik Bouaziz

June 17, 2014

TR-97

A TimeCenter Technical Report

Title	An Efficient Approach for Detecting and Repairing Data Inconsistencies Resulting from Retroactive Updates in Multi-temporal and Multi-version XML Databases
Author(s)	Hind Hamrouni, Zouhaier Brahmia, Rafik Bouaziz
Publication History	June 2014. A TimeCenter Technical Report.

TimeCenter Participants

Michael H. Böhlen, University of Zurich, Switzerland; Curtis E. Dyreson, Utah State University, USA; Fabio Grandi, University of Bologna, Italy; Christian S. Jensen (codirector), Aarhus University, Denmark; Vijay Khatri, Indiana University, USA; Gerhard Knolmayer, University of Berne, Switzerland; Carme Martín, Technical University of Catalonia, Spain; Thomas Myrach, University of Berne, Switzerland; Mario A. Nascimento, University of Alberta, Canada; Sudha Ram, University of Arizona, USA; John F. Roddick, Flinders University, Australia; Keun H. Ryu, Chungbuk National University, Korea; Simonas Šaltenis, Aalborg University, Denmark; Dennis Shasha, New York University, USA; Richard T. Snodgrass (codirector), University of Arizona, USA; Paolo Terenziani, University of Piemonte Orientale "Amedeo Avogadro," Alessandria, Italy; Stephen W. Thomas, Queen's University, Canada; Kristian Torp, Aalborg University, Denmark; Vassilis Tsotras, University of California, Riverside, USA; Fusheng Wang, Emory University, USA; Jef Wijsen, University of Mons-Hainaut, Belgium; and Carlo Zaniolo, University of California, Los Angeles, USA

For additional information, see The TimeCenter Homepage:
URL: `<http://www.cs.aau.dk/TimeCenter>`

The TimeCenter icon on the cover combines two "arrows." These "arrows" are letters in the so-called *Rune* alphabet used one millennium ago by the Vikings, as well as by their precedessors and successors. The Rune alphabet (second phase) has 16 letters, all of which have angular shapes and lack horizontal lines because the primary storage medium was wood. Runes may also be found on jewelry, tools, and weapons and were perceived by many as having magic, hidden powers.

The two Rune arrows in the icon denote "T" and "C," respectively.

An Efficient Approach for Detecting and Repairing Data Inconsistencies Resulting from Retroactive Updates in Multi-temporal and Multi-version XML Databases

Hind Hamrouni, Zouhaier Brahmia, Rafik Bouaziz

University of Sfax, Tunisia

emails : hindhamrouni@gmail.com, {zouhaier.brahmia, raf.bouaziz}@fsegs.rnu.tn

Abstract

Multi-temporal XML databases supporting schema versioning contain XML elements of different temporal formats (snapshot, transaction-time, valid-time, and bitemporal), defined under several XML schema versions. These databases support three types of data updates concerned with the time when updates are made: retroactive, proactive, or on-time, dealing with past, future, or current data respectively. A retroactive update (i.e., modifying or deleting a past element) due to a detected error means that the database has included erroneous information during some period in the past and, therefore, its consistency should be restored by correcting all errors and inconsistencies that have occurred in the past. Indeed, all processings that have been carried out during the inconsistency period and have used erroneous information have normally produced erroneous information. In this work, we propose an approach which preserves data consistency in multi-temporal and multi-version XML databases. More precisely, after any retroactive update, the proposed approach allows (i) detecting and analyzing periods of database inconsistency, which result from that update, and (ii) repairing of all inconsistencies and recovery of all side effects.

Keywords: Temporal XML Databases; Schema Versioning; Retroactive Update; Data Inconsistency; Inconsistency Periods; Repairing an Inconsistency; Side Effect; Side Effects Recovery.

1 Introduction

Nowadays, supporting the temporal aspect is a requirement for most computer applications, including medical, legal, banking, scientific and scheduling applications. In fact, these applications need to store and manipulate data while taking into account the time dimension. This has led to the appearance of temporal databases [1, 2, 3] which retain data evolution over transaction-time dimension (concerning the real world) and/or valid-time dimension (concerning the database life) [4]:

- The valid-time of a datum is the time when a datum is true in the real world; each time-varying data is timestamped with a validity start time (VST) and a validity end time (VET).
- The transaction-time of a datum is the time when a datum is current in the database; each time-varying data is timestamped with a transaction start time (TST) and a transaction end time (TET).

Thus, according to the temporal dimensions they support, temporal databases are classified into five categories [4, 5]: transaction-time (including only transaction-time data), valid-time (containing only valid-time data), bitemporal (storing only bitemporal data), snapshot (containing only non-temporal/conventional data) or multi-temporal (supporting data of different temporal formats).

Besides, evolution of database schema (e.g., dropping or adding entities, dropping or adding attributes of entities) over time is unavoidable in the context of information systems and may occur due to several reasons: meeting new user requirements, taking into account new regulations, optimizing the database schema in order to have best performance, etc. In temporal databases, changing schema could lead to some problems like data loss when dropping some entities and/or attributes. Thus, although temporal databases allow keeping track of data history when the schema is static, they do not provide a complete history when they do not keep track also of database schema evolution over time. Therefore,

to avoid this drawback and to provide a complete data history which allows performing operations on data defined under any database schema version, researchers in the database community have proposed to adopt the schema versioning technique in temporal databases [5, 6, 7].

In multi-temporal databases, there are three types of updates concerned with the time when updates are made: retroactive, proactive [8], and real-time (or on-time) updates.

- A retroactive update is performed after the change occurred in reality (i.e., the TST of the datum is superior to its VST).
- A proactive update is done before the change occurs in reality (i.e., the TST of the datum is inferior to its VST).
- A real-time update is done when the change occurs in reality (i.e., the TST of the datum is equal to its VST).

Retroactive and proactive updates [8] occur naturally in many applications. For example, a postdated check is a proactive update, and a salary increase may be retroactive to some past date.

On the other hand, currently, XML databases [9] are widely used, especially on the Web. The introduction of temporal [10] aspects in such databases gave rise to temporal XML databases [11]. In these databases, any temporal XML document can store transaction-time, valid-time and bi-temporal XML elements. Moreover, these databases are very useful for several domains (e.g., managing evolution of e-commerce customer profiles, managing evolution of legal texts in e-government systems, online management of patients' medical records...). Notice that temporal XML databases are richer than temporal relational databases at structure and textual content levels. Moreover, temporal XML data are presented with temporally grouped data models [12], which have long been advocated as the most natural and effective representations of temporal information [13]. Indeed, each time-varying XML element evolves individually over time.

Under such databases, end users/applications can insert, delete, and modify past, current and future data. However, these updates are not always performed safely (i.e., without problems), since sometimes data update operations can have a harmful effect on the consistency of the database. Indeed, compromising the database consistency could occur when manipulating any type of data (i.e., current, past, or future). In the following, we better explain this issue through three cases of data manipulation operations (we suppose that the current time is 2014-04-01):

Case 1: Detecting database inconsistency when manipulating current data.

Let's take the example of correcting the current XML element representing the current price of a product, which was introduced on 2014-02-01. All existing data (e.g., monthly turnovers, total amounts of customer invoices) that have been produced using the erroneous price are consequently erroneous and should be corrected. The database has included inconsistencies during the period going from 2014-02-01 to 2014-04-01.

Case 2: Detecting database inconsistency when manipulating past data.

Let's take the sample of correcting a past XML element representing a past banking interest rate in a bank, which was applied during the period going from 2013-01-01 to 2013-12-31. All existing data (e.g., interest bank accounts, balances of bank accounts, scheduled payment amounts) that have been obtained using the erroneous past banking interest rate are consequently erroneous and should be corrected. The database was inconsistent during the period where this interest rate was effective.

Case 3: Detecting database inconsistency when manipulating future data.

Let's take the example of correcting a future XML element representing the amount of projected annual sales of a company, which was introduced in the database on 2014-01-10. All existing data (e.g., future statistics, economic forecasts) that have been calculated using the erroneous future amount are consequently erroneous and should be corrected. The database was in an inconsistent state between 2014-01-10 and 2014-04-01.

In the present work, we deal only with the second case; we left the two other cases to future works. We focus on the impacts of manipulating past data on the consistency of the database: we show how to detect inconsistencies and to repair the database consistency, after a transaction which includes some operations acting on past bitemporal or valid-time data.

For the best of our knowledge, existing database management systems (DBMSs), temporal integrity constraints [14] and semantics of existing temporal data manipulation operations [15, 16] are insufficient to cope with such problems, since they do not provide any support to ensure the database consistency in cases like the three ones presented above. In fact, although some existing DBMSs, like Oracle 12c, DB2 ver.10, and Teradata Database, include some features for temporal data types and management [3], none of them provides support for determining database consistency periods and repairing consistencies resulting from retroactive updates. As for available temporal integrity constraints, they only guarantee that some rules on data (e.g., key, uniqueness) are being respected and some relations between data are established (e.g., referential integrity constraint). Moreover, the previously proposed temporal data manipulation operations [15, 16] only perform actions on temporal data according to a temporal semantics which does not take into account the checking of database consistency, since they neither inform the end-user that an inconsistency period has been detected, nor activate supplementary and transparent processings in order to restore the database consistency in cases similar to what presented above.

To solve this problem, we neither add new temporal integrity constraints, nor update the current semantics of temporal data manipulation operations. We propose a new approach that allows the temporal XML database management system (i) to detect any database inconsistency that happens after a retroactive data manipulation operation, and (ii) to perform necessary processings, in a transparent way, in order to repair inconsistencies.

The rest of this paper is organized as follows: the next section motivates the need for a new approach for preserving the consistency of a multi-temporal XML database supporting schema versioning; Section 3 describes data inconsistencies resulting from retroactive updates in such a database; Section 4 presents our approach for automatic repairing of data inconsistencies that occur due to a retroactive update; Section 5 discusses related work; Section 6 concludes the report with a summary and a list of our future work.

2 Motivation

In this section, we first present two examples that illustrate how maintaining consistency in multi-temporal and multi-version XML databases after a retroactive update is a complicated task that could not be achieved using existing practical supports, i.e., supports provided by DBMSs. Then we show the need for systems providing supports for preserving consistency of muti-temporal and multi-version XML databases.

2.1 Motivating Examples

In order to show that retroactive updates are not graceful tasks, we present the following two examples:

Example 1: Data inconsistencies resulting from a retroactive update of an employee's salary

Suppose that on 2014-06-10, the personnel officer detects an error that has occurred on 2012-01-05: he/she saved an erroneous value for the salary of the employee Abdullah: 1120 TND (the erroneous value) instead of 1210 TND (the correct value); thus, an amount of 90 TND has not been considered in the salary, and during a period of twenty-nine months. Obviously, this error was propagated to all results of operations that have been performed using this salary (i.e., 1120), especially those which calculated taxes and social security contributions, as well as to all successor salaries.

Currently, the semantics of the temporal data update operations [15, 16], which should be used to correct the erroneous salary that was inserted on 2012-01-05, does not support the correction of all

effects of this error (i.e., it does not correct all taxes and social security contributions, that were calculated after 2012-01-05 based on the erroneous salary, as well as all other successor salaries). Such an operation corrects only the value of the corresponding salary. The XML element that represents the erroneous salary is stored as a past erroneous element, and the correct salary is stored in a new XML element which represents a past correct element <salary/> for that employee (see Fig. 1); the modification is performed in a non-destructive manner, since we are in a temporal setting.

```
<employees>
 <employee>
  <SSN>12345678</SSN>
  <name>Abdullah</name>
  <salaries>
   <salary  VST="2012-01-01"  VET="2012-12-31"
            TST="2012-01-05"  TET="2013-01-03">1120</salary>
   <salary  VST="2012-01-01"  VET="2012-12-31"
            TST="2014-06-10"  TET="2014-06-10">1210</salary>
   <salary  VST="2013-01-01"  VET="2013-12-31"
            TST="2013-01-03"  TET="2014-01-03">1250</salary>
   <salary  VST="2013-01-01"  VET="2013-12-31"
            TST="2014-06-10"  TET="2014-06-10">1300</salary>
   <salary  VST="2014-01-01"  VET="2014-12-31"
            TST="2014-01-03"  TET="2014-06-10">1390</salary>
   <salary  VST="2014-01-01"  VET="2014-12-31"
            TST="2014-06-10"  TET="UC">1530</salary>
  </salaries>
  <socialSecurityContributions>
   <socialContribution  VST="2012-01-01"  VET="2012-12-31"
            TST="2012-01-05"  TET="2013-01-03">112</socialContribution>
   <socialContribution  VST="2012-01-01"  VET="2012-12-31"
            TST="2014-06-10"  TET="2014-06-10">121</socialContribution>
   <socialContribution  VST="2013-01-01"  VET="2013-12-31"
            TST="2013-01-03"  TET="2014-01-03">125</socialContribution>
   <socialContribution  VST="2013-01-01"  VET="2013-12-31"
            TST="2014-06-10"  TET="2014-06-10">130</socialContribution>
   <socialContribution  VST="2014-01-01"  VET="2014-12-31"
            TST="2014-01-03"  TET="2014-06-10">139</socialContribution>
   <socialContribution  VST="2014-01-01"  VET="2014-12-31"
            TST="2014-06-10"  TET="UC">153</socialContribution>
  </socialSecurityContributions>
  <taxes>
   <tax    VST="2012-01-01"  VET="2012-12-31"
           TST="2012-01-05"  TET="2013-01-03">56</tax>
   <tax    VST="2012-01-01"  VET="2012-12-31"
           TST="2014-06-10"  TET="2014-06-10">60.5</tax>
   <tax    VST="2013-01-01"  VET="2013-12-31"
           TST="2013-01-03"  TET="2014-01-03">62.5</tax>
   <tax    VST="2013-01-01"  VET="2013-12-31"
           TST="2014-06-10"  TET="2014-06-10">65</tax>
   <tax    VST="2014-01-01"  VET="2014-12-31"
           TST="2014-01-03"  TET="2014-06-10">69.5</tax>
   <tax    VST="2014-01-01"  VET="2014-12-31"
           TST="2014-06-10"  TET="UC">76.5</tax>
  </taxes>
 </employee>
 …
<employees>
```

Fig. 1. Salaries of the employee Abdullah, and corresponding social security contributions and taxes, corrected after a retroactive update.

To repair all inconsistencies that are stored in the database, the DBA should proceed in an ad hoc manner: first, he/she should determine the list of all operations that were done using the erroneous salary, in order to define all data that were calculated based on the erroneous salary or calculated based on other data obtained from the erroneous salary, going from 2012-01-05 to 2014-06-10. Then, he/she should correct all erroneous data, either manually or by writing an appropriate XML update [16, 17, 18]. Fig. 1 shows that all salaries introduced after the corrected salary have been also corrected by inserting new correct past <salary/> elements, except the last salary which is replaced by a new correct current <salary/> element (i.e., the <salary/> element with TET attribute equal to "UC"); notice that "UC" (Until Change) [4] means that the salary is current until new change. The figure 1 shows also that both the social security contributions and taxes corresponding to the corrected salaries have also been corrected accordingly (a social security contribution is equal to 10% of the salary paid during the same period, whereas a tax is equal to 5% of the corresponding salary).

Example 2: Data inconsistencies resulting from a retroactive update of a deposited amount in a bank account

Suppose that on 2014-03-25, the auditor detects an error that has occurred on 2014-01-05: the bank employee saved a deposit transaction which adds an amount of 550 TND (the erroneous value) instead of 500 TND (the correct value), to the account of a customer; thus, an amount of 50 TND was stored in the database but really was not provided by the customer. Obviously, this error was propagated to all other financial transactions that have been done on this account between 2014-01-05 and 2014-03-25; there was always an amount of 50 TND which should be subtracted from the balance.

As said in Example 1, the semantics of existing data update operation [15, 16], which should be used to correct both the deposited amount and the balance account on 2014-01-05, does not support the correction of the impact of this error (i.e., to correct each balance of this account, related to each transaction performed after 2014-01-05). Such an operation corrects only the details of the financial transaction (i.e., the deposited amount and the balance of the account at the end of the transaction) done on 2014-01-05. To restore the database consistency, the DBA should proceed in an ad hoc manner: first, he/she should determine the list of all transactions that were done this account going from the transaction during which the error has occurred until the last one. Then, he/she should update all erroneous data by writing an appropriate XML update query or a program.

2.2 Need for New DBMS Supports

The consistency of a multi-temporal and multi-version XML database could not be ensured easily, since (i) all temporal dimensions are supported (i.e., data can evolve over transaction time and/or valid time), (ii) consistency of both conventional and temporal data should be guaranteed, (iii) there are several schema versions and consequently several database instances, and (iv) a data management operation that is originally devoted to insert, delete, or update an XML element could involve several other XML elements (e.g., when the temporal interval of a new element that modifies an existing one overlaps, completely or partially, temporal intervals of other existing XML elements), defined under the same XML schema version or under several XML schema versions.

Thus, the challenges described above show that end users/applications, interacting with muti-temporal and multi-version XML databases, need DBMSs with built-in support for automatic and graceful repairing of data inconsistencies that result from retroactive updates.

2 Data Inconsistencies Resulting from Retroactive Updates

Inserting, updating and deleting past data can give rise to inconsistencies in multi-temporal and multi-version XML databases. These inconsistencies generate many side effects affecting other data.

In [19], the author defined a side effect as a database inconsistency generated by a processing which has used an inconsistent data. A side effect is generated by (i) a retroactive update of data, (ii) a cancellation of effects of a previous processing, or (iii) a re-execution of a previous processing with

correct values for all written and read data (in fact, such a processing had used during its (first) execution erroneous values for some written or read data, and, thus, it should be re-executed in order to repair inconsistencies that have occurred). Moreover, the author of [19] defined an inconsistency period resulting from a retroactive update of data as the temporal interval which delimits the scope of side effects that are expected to be generated by this update. Such an inconsistency period could be one of the following three types: "Wrong Absence of Data", "Wrong Presence of Data", or "Errors in Data".

- Wrong Absence of Data: the inconsistency is due to the absence of a datum that had to be present in the database during this period;
- Wrong Presence of Data: the inconsistency comes from the presence of a datum that had to be absent in the database during this period;
- Errors in Data: the inconsistency results from the existence of some data with erroneous values during this period.

An inconsistency period, resulting from a retroactive update can be divided into several sub-periods; each one of these sub-periods should be interpreted according to the nature (i.e., insertion, deletion, or correction) of the retroactive update. In the following, we study periods of inconsistency resulting from retroactive updates, and their sub-periods.

3.1 Data inconsistencies resulting from a retroactive insertion of data

The insertion of an XML element with retroactive effect generates an inconsistency period of "Wrong Absence of Data" type: the inserted element, which was absent before its transaction start time, should be normally present in the temporal database since its validity start time. Fig. 2 illustrates such a period of inconsistency. In the following, we use CT to denote the "Current Time".

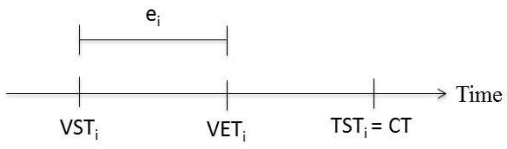

Fig. 2. The inconsistency period resulting from a retroactive insertion of data.

As shown by Fig. 2, the period of inconsistency, which is delimited by the VST (period beginning) and the TST (period ending) of e_i, can be divided into two sub-periods:

- $[VST_i – VET_i]$: the interval during which the consequent side effects concern all processings that had to use the element e_i while it had to be a current element;
- $]VET_i – TST_i]$: the interval during which the generated side effects concern all processings that had to use the element e_i while it had to be a past element.

3.2 Data Inconsistencies Resulting from a Retroactive Deletion of Data

The removal of an XML element with retroactive effect generates an inconsistency period of "Wrong Presence of Data" type: the deleted element, which was present before the instant of its deletion (i.e., before the transaction start time of the deletion element [15, 16], that is the XML element which is used to delete the corresponding element), should not normally exist in the temporal database since its validity start time. Fig. 3 illustrates such a period of inconsistency.

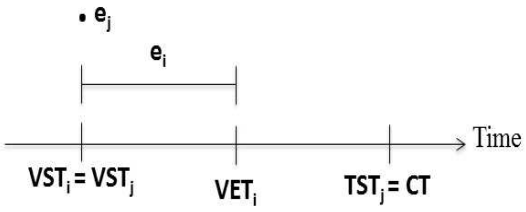

Fig. 3. The inconsistency period resulting from a retroactive deletion of data.

As shown by Fig. 3, the period of inconsistency, which is delimited by the VST of e_i (period beginning) and the TST of e_j (period ending), can be divided into two sub-periods:

- $[VST_j - VET_i]$: the interval during which the resulting side effects concern all processings that had used the element e_i while it was a current element;
- $]VET_i - TST_j]$: the interval during which the consequent side effects concern all processings that had used the element e_i while it was a past element.

3.3 Data Inconsistencies Resulting from a Retroactive Correction of Data

Retroactive correction operations could be done only on valid-time and bitemporal data. In this paper, we deal only with retroactive correction of bitemporal data, since we think that it is more complex and, thus, it requires much attention.

The correction of a bitemporal element is performed by inserting a new element containing the correct values, called the element of correction [15, 16]. In the following examples, e_j denotes the correction element and e_i denotes the corrected element. A correction operation can affect (i) the contents of the corrected element, (ii) values of non-temporal attributes (i.e., attributes different of VST, VET, TST, and TET attributes) of the corrected element, and/or (iii) the valid-time interval of the corrected element (i.e., values of VST and VET attributes); obviously, the transaction-time interval (i.e., values of TST and TET attributes) of any element cannot be modified owing to the definition of transaction time. In the first and/or the second case (i.e., points (i) and (ii)), the correction operation generates an inconsistency period of type "Errors in Data" (see the following first subsection). However, in the third case (i.e., point (iii)), it generates an inconsistency period which can be divided into several sub-periods each one of them has a different type (see the following second subsection). In the following two subsections, we study these inconsistencies.

3.3.1 Data inconsistencies resulting from a retroactive correction operation which does not modify the valid-time interval of a bitemporal element

In this case, the retroactive correction operation updates the contents and/or the values of non-temporal attributes of a bitemporal element; it modifies neither the VST attribute, nor the VET attribute. This correction generates an inconsistency period of type "Errors in Data": it means that the corrected element had an erroneous value. Fig. 4 illustrates such a period of inconsistency.

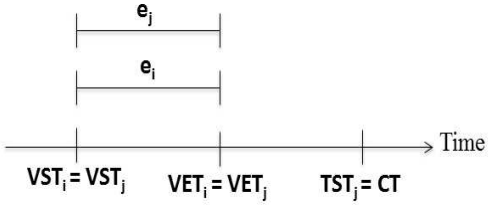

Fig. 4. The inconsistency period resulting from a retroactive correction operation which does not modify the valid-time interval of a bitemporal element.

As shown by Fig. 4, the period of inconsistency, which is delimited by the VST of e_i (period beginning) and the TST of e_j (period ending), can be divided into two sub-periods:

- $[VST_i - VET_i]$: an inconsistency period of type "Errors in Data": the interval during which the generated side effects concern all processings that had used the element e_i while it was a current element;
- $]VET_i - TST_j]$: an inconsistency period of type "Errors in Data": the interval during which the consequent side effects concern all processings that had used the element e_i while it was a past element.

3.3.2 Data inconsistencies resulting from a retroactive correction operation which modifies the valid-time interval of a bitemporal element

In this subsection, we study inconsistencies resulting from a retroactive correction operation that modifies the VST attribute and/or the VET attribute. When the valid-time interval of a bitemporal element is modified, we distinguish different cases according to Allen's interval algebra [20] and all possible relations between the valid-time interval of the correction element (e_j) and that of the corrected element (e_i).

Case 1: The valid-time interval of the correction element (e_j) follows that of the corrected element (e_i)

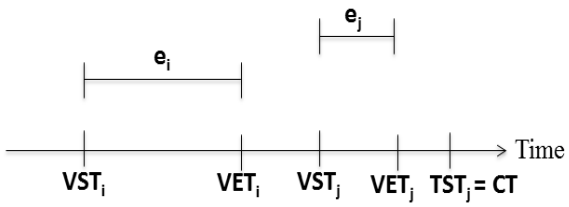

Fig. 5. The inconsistency period resulting from a retroactive correction operation which replaces the valid-time interval of a bitemporal element with a new one that follows it.

As shown by Fig. 5, the period of inconsistency, which is delimited by the VST of e_i (period beginning) and the TST of e_j (period ending), can be divided into four sub-periods:

- $[VST_i – VET_i]$: an inconsistency sub-period of type "Wrong Presence of Data", during which the consequent side effects concern all processings that had used e_i while it was a current element;
- $]VET_i - VST_j[$: an inconsistency sub-period of type "Wrong Presence of Data", during which the generated side effects concern all processings that had used e_i while it was a past element;
- $[VST_j - VET_j]$:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the generated side effects concern all processings that had to use e_j while it had to be a current element;
  - it could also be considered as an inconsistency sub-period of type "Errors in Data" only and only if the correction element e_j updates also the contents and/or at least one of the non-temporal attributes of the corrected element e_i. So, during this sub-period, the resulting side effects concern all processings that had used e_i while it was a past element.
- $]VET_j - TST_j]$:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use e_j while it had to be a past element;

- it could also be considered as an inconsistency sub-period of type "Errors in Data" only and only if the [VST_j – VET_j] sub-period has also this type. In such a case, during this sub-period, the resulting side effects concern all processings that had used e_i while it was a past element.

Case 2: The valid-time interval of the correction element (e_j) contains that of the corrected element (e_i)

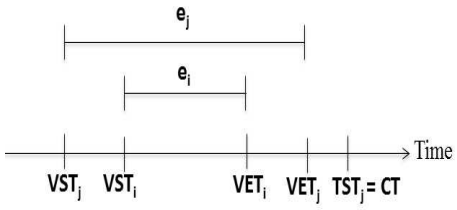

Fig. 6. The inconsistency period resulting from a retroactive correction operation which replaces the valid-time interval of a bitemporal element with a new one that contains it.

As shown by Fig. 6, the period of inconsistency, which is delimited by the VST of e_j (period beginning) and the TST of e_j (period ending), can be divided into four sub-periods:

- [VST_j – VST_i[: an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use the element e_j while it had to be a current element;
- [VST_i – VET_i]: this interval is not considered always as an inconsistency sub-period; it could be considered as an inconsistency sub-period of type "Errors in Data" only and only if the correction element e_j updates also the contents and/or at least one of the non-temporal (i.e., non-timestamping) attributes of the corrected element e_i. In that case, during [VST_i – VET_i], the generated side effects concern all processings that had used the element e_i while it was a current element.
-]VET_i – VET_j]:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use e_j while it had to be a current element;
  - it also could be considered as an inconsistency sub-period of type "Errors in Data" only and only if the [VST_i – VET_i] sub-period has also this type. In that case, the resulting side effects during this sub-period concern all processings that had used e_i while it was a past element.
-]VET_j - TST_j]:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use e_j while it had to be a past element;
  - it could also be considered as an inconsistency sub-period of type "Errors in Data" only and only if the [VST_i – VET_i] sub-period has also this type. In such a case, the side effects generated during this sub-period concern all processings that had used e_i while it was a past element.

Case 3: The valid-time interval of the correction element (e_j) precedes that of the corrected element (e_i)

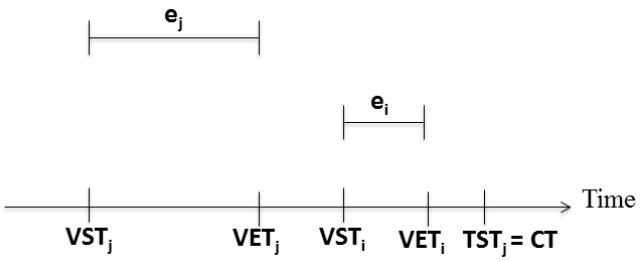

Fig. 7. The inconsistency period resulting from a retroactive correction operation which replaces the valid-time interval of a bitemporal element with a new one that precedes it.

As shown by Fig. 7, the period of inconsistency, which is delimited by the VST of e_j (period beginning) and the TST of e_j (period ending), can be divided into four sub-periods:

- $[VST_j – VET_j[$: an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use the element e_j while it had to be a current element;
- $[VET_j – VST_i]$: an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use the element e_j while it had to be a current element;
- $]VST_j – VET_i]$:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use e_j while it had to be a past element;
  - it also could be considered as an inconsistency sub-period of type "Errors in Data" only and only if the $[VST_i – VET_i]$ sub-period has also this type. In that case, the resulting side effects during this sub-period concern all processings that had used e_i while it was a current element.
- $]VET_j - TST_j]$:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use e_j while it had to be a past element;
  - it could also be considered as an inconsistency sub-period of type "Errors in Data" only and only if the $[VST_i – VET_i]$ sub-period has also this type. In such a case, the side effects generated during this sub-period concern all processings that had used e_i while it was a past element.

Case 4: The valid-time interval of the correction element (e_j) included in that of the corrected element (e_i)

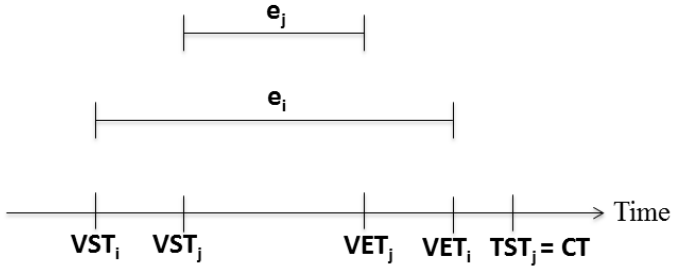

Fig. 8. The inconsistency period resulting from a retroactive correction operation which replaces the valid-time interval of a bitemporal element with a new one that included in it.

As shown by Fig. 8, the period of inconsistency, which is delimited by the VST of e_j (period beginning) and the TST of e_j (period ending), can be divided into four sub-periods:

- $[VST_i – VST_j[$: an inconsistency sub-period of type "Wrong Presence of Data", during which the consequent side effects concern all processings that had used the element e_j while it was a current element;
- $[VST_j – VET_j]$: this interval is not considered always as an inconsistency sub-period; it could be considered as an inconsistency sub-period of type "Errors in Data" only and only if the correction element e_j updates also the contents and/or at least one of the non-temporal (i.e., non-timestamping) attributes of the corrected element e_i. In that case, during $[VST_j – VET_j]$, the generated side effects concern all processings that had used the element e_i while it was a current element.
- $]VET_j – VET_i]$:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use e_j while it had to be a past element;
  - it also could be considered as an inconsistency sub-period of type "Errors in Data" only and only if the $[VST_j – VET_j]$ sub-period has also this type. In that case, the resulting side effects during this sub-period concern all processings that had used e_i while it was a current element.
- $]VET_i - TST_j]$:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use e_j while it had to be a past element;
  - it could also be considered as an inconsistency sub-period of type "Errors in Data" only and only if the $[VST_j – VET_j]$ sub-period has also this type. In such a case, the side effects generated during this sub-period concern all processings that had used e_i while it was a past element.

Case 5: The valid-time interval of the correction element (e_j) overlaps that of the corrected element (e_i) on right

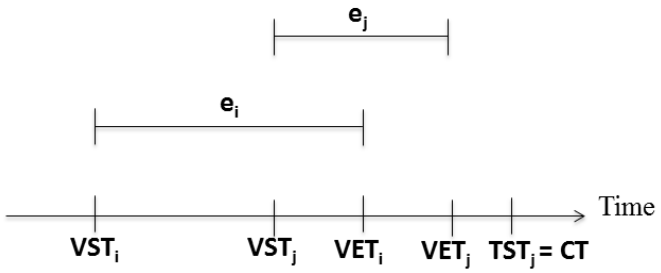

Fig. 9. The inconsistency period resulting from a retroactive correction operation which replaces the valid-time interval of a bitemporal element with a new one that overlaps it on the right.

As shown by Fig. 9, the period of inconsistency, which is delimited by the VST of e_j (period beginning) and the TST of e_j (period ending), can be divided into four sub-periods:

- $[VST_i – VST_j[$: an inconsistency sub-period of type "Wrong Presence of Data", during which the consequent side effects concern all processings that had used the element e_j while it had to be a current element;
- $[VST_j – VET_i]$: this interval is not considered always as an inconsistency sub-period; it could be considered as an inconsistency sub-period of type "Errors in Data" only and only if the

correction element e_j updates also the contents and/or at least one of the non-temporal (i.e., non-timestamping) attributes of the corrected element e_i. In that case, during $[VST_j - VET_i]$, the generated side effects concern all processings that had used the element e_i while it was a current element.

- $]VET_i - VET_j]$:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use e_j while it had to be a current element;
  - it also could be considered as an inconsistency sub-period of type "Errors in Data" only and only if the $[VST_j - VET_i]$ sub-period has also this type. In that case, the resulting side effects during this sub-period concern all processings that had used e_i while it was a past element.
- $]VET_j - TST_j]$:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use e_j while it had to be a past element;
  - it could also be considered as an inconsistency sub-period of type "Errors in Data" only and only if the $[VST_i - VET_i]$ sub-period has also this type. In such a case, the side effects generated during this sub-period concern all processings that had used e_i while it was a past element.

Case 6: The valid-time interval of the correction element (e_j) overlaps that of the corrected element (e_i) on left

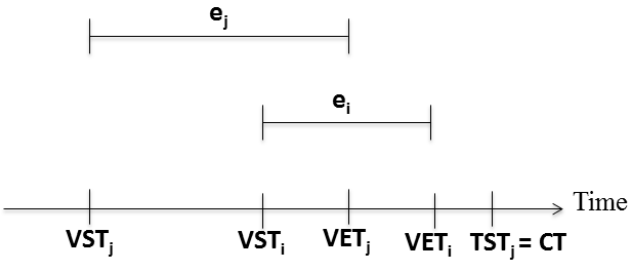

Fig. 10. The inconsistency period resulting from a retroactive correction operation which replaces the valid-time interval of a bitemporal element with a new one that overlaps it on left.

As shown by Fig. 10, the period of inconsistency, which is delimited by the VST of e_j (period beginning) and the TST of e_j (period ending), can be divided into four sub-periods:

- $[VST_j - VST_i[$: an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use the element e_j while it had to be a current element;
- $[VST_i - VET_j]$: this interval is not considered always as an inconsistency sub-period; it could be considered as an inconsistency sub-period of type "Errors in Data" only and only if the correction element e_j updates also the contents and/or at least one of the non-temporal (i.e., non-timestamping) attributes of the corrected element e_i. In that case, during $[VST_i - VET_j]$, the generated side effects concern all processings that had used the element e_i while it was a current element.
- $]VET_j - VET_i]$:
  - this sub-period is considered as an inconsistency sub-period of type "Wrong Absence of Data", during which the consequent side effects concern all processings that had to use e_j while it had to be a current element;

- it also could be considered as an inconsistency sub-period of type “Errors in Data” only and only if the [VST_i – VET_j] sub-period has also this type. In that case, the resulting side effects during this sub-period concern all processings that had used e_i while it was a past element.

♦]VET_j - TST_j]:

- this sub-period is considered as an inconsistency sub-period of type “Wrong Absence of Data”, during which the consequent side effects concern all processings that had to use e_j while it had to be a past element;
- it could also be considered as an inconsistency sub-period of type “Errors in Data” only and only if the [VST_i – VET_j] sub-period has also this type. In such a case, the side effects generated during this sub-period concern all processings that had used e_i while it was a past element.

4 Detecting and Repairing Data Inconsistencies Resulting from Retroactive Updates in Multi-temporal XML Databases Supporting Schema Versioning

In this section, we propose an approach that allows restoring automatically the multi-temporal database consistency after a retroactive update of temporal XML data. First, we describe the process of restoring automatically the database consistency. Then, we present the architecture of a native multi-temporal and multi-version XML DBMS which supports restoring automatically the database consistency after a retroactive update.

4.1 Process of Restoring Automatically the Database Consistency

When an end user or an application submits to the temporal XML DBMS a retroactive update of temporal XML data, the DBMS performs the following sequence of tasks:

Task 1: it updates the database as required by the end user or the application (obviously after checking the update syntactically).

Task 2: it determines automatically the period of inconsistency resulting from the retroactive update of data, and its sub-periods.

Task 3: it determines automatically the list of transactions that were executed during each sub-period of inconsistency and had used erroneous past data (in case that the corresponding sub-period of inconsistency is of type “Wrong Presence of Data” or “Errors in Data”) or had to use new data (in case that the corresponding sub-period of inconsistency is of type “Wrong Presence of Data” or “Errors in Data”); for each concerned transaction, it should provide its commit time, all its elementary operations (for the sake of simplicity, we suppose that a transaction is composed of a single operation, i.e., a single insert, delete, or update operation), and all data that were written and read by this transaction.

Task 4: it re-executes in a provisory workspace the list of corresponding transactions during each sub-period of inconsistency either (a) without using the corresponding past data, if this sub-period is of type “Wrong Presence of Data”, or (b) while using (b.1) the correct values of past data, if this sub-period is of type “Errors in Data”, or (b.2) the specified past data, if this sub-period is of type “Wrong Absence of Data”.

Task 5: it compares the old results of determined transactions (i.e., results already stored in the database, as written data by these transactions) with the new results of them (i.e., the new data that are written by these transactions in the provisory workspace).

Task 6: it replaces every old result with the corresponding new result when there is a difference between them.

In the following, we provide main requirements of some tasks presented above:

- Task 1 requires that the DBMS supports management of temporal XML data under schema versioning; we have studied this aspect in our previous work [15, 16].
- Task 3 requires beforehand keeping track of all transactions which are executed: all operations which compose each transaction, all written and read data, and its commit time;
- Task 4 requires that the provisory workspace should be a copy of the database (schema and instances) during the inconsistency period, before the execution of the retroactive update operation.
- Task 6 requires that replacing old data with new data should be performed logically and not physically (i.e., in a destructive manner); each existing erroneous data is logically corrected by a new correct data. After restoring the database consistency, only correct data must be used by the DBMS to answer user/application queries; erroneous data could be vacuumed later [21] by the database administrator.

4.2 Architecture

Owing to the general architecture of a DBMS [22], the transaction manager is the component which is devoted to managing transactions resulting from user/applications queries and updates submitted to the database. Therefore, if we would like to have an automatic restoring of the database consistency after any retroactive update of temporal XML data, we think that (i) the transaction manager of a temporal XML DBMS should be extended by four new components: "Retroactive Update Checker", "Inconsistency Period Manager", "Side Effect Recovery Manager" and "Optimizer", and (ii) the temporal XML DBMS itself should include a "Transaction Catalog Manager", a "Transaction Catalog", and a "Provisory Workspace". The new general architecture of a temporal XML DBMS is depicted in Fig. 11.

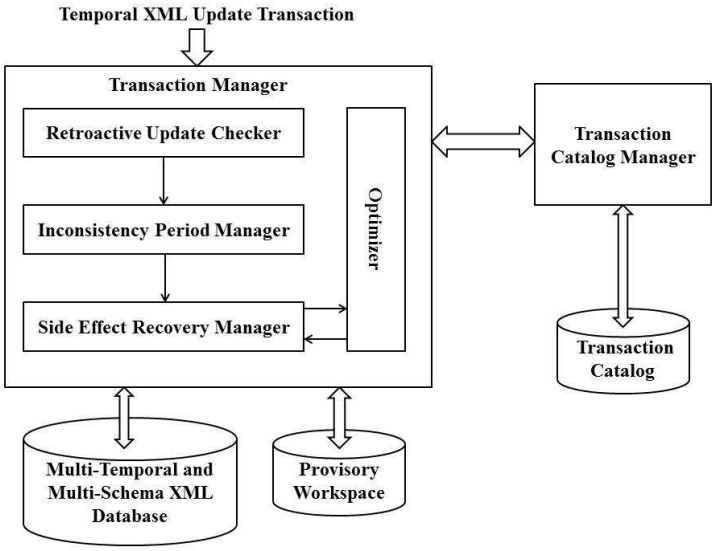

Fig. 11. General Architecture of a temporal XML database supporting automatic detecting and repairing of data inconsistencies resulting from retroactive updates.

The "**Retroactive Update Checker**" checks that the Temporal XML Update submitted by the end user/application is an update operation with retroactive effect (i.e., this operations adds, deletes, or modifies past data). Notice that a past data has a valid-time interval which ends before the current time (i.e., VET < CT).

The "**Inconsistency Period Manager**", which is invoked by the "**Retroactive Update Checker**" in case it detects a retroactive update, determines automatically the period of inconsistency which results

from a retroactive update of data, and its sub-periods with their types. Suppose that an erroneous past element "ee" is corrected by a correct past element "ce". The resulting period of inconsistency is defined by either the interval [VST_{ee} - CT] (if $VST_{ee} < VST_{ce}$), or [VST_{ce} - CT] (if $VST_{ce} < VST_{ee}$). The sub-periods of inconsistency related to this period are identified according to the study presented in the subsection 3.3.

After determining all sub-periods of inconsistency, the "**Inconsistency Period Manager**" (i) invokes the "**Transaction Catalog Manager**" in order to retrieve from the "Transaction Catalog" the list of transactions that were executed during each one of these sub-periods, and (ii) sends all retrieved transactions to the "**Side Effect Recovery Manager**".

The "**Side Effect Recovery Manager**" controls the re-execution of transactions which have used erroneous data (i.e., transactions executed during a period of inconsistency of type "Wrong Presence of Data" or "Errors in Data") and transactions that had to use newly added data (i.e., transactions executed during a period of inconsistency of type "Wrong Absence of Data"). The concerned transactions are re-executed in a "Provisory Workspace" which is a copy of the database during the corresponding period of inconsistency. At the end of the re-execution of each transaction, the "**Side Effect Recovery Manager**" compares the results of determined transaction already stored (as written data) in the database with the results of the execution in the provisory workspace. If there are differences between them, so an inconsistency is detected and it should be repaired. For that, the "**Side Effect Recovery Manager**" replaces the old result with the new one.

The "**Side Effect Recovery Manager**" interacts with the "**Optimizer**" module which implements a set of optimization rules. Indeed, the "**Optimizer**" receives a sequence of non-optimized transactions that should be re-executed, and tries to reduce them (if possible). In the following, we present three examples of these optimization rules:

- ***Rule 1:*** if two successive transactions, T1 and T2, act on the same XML element e12, such that T1 adds e12 and T2 deletes e12, then the system has to ignore these two transactions and do not re-execute them.
- ***Rule 2:*** if two successive transactions, T3 and T4, act on the same XML element e34, such that T3 adds e34 and T4 updates e34, then the system has to combine/merge the two transactions into the first one (which is T3) and re-executing it (i.e., re-executing T3) but with updated values provided in the second one (which is T4): adding e34 with updated values provided in T4.
- ***Rule 3:*** if a transaction does not include any operation of type data insertion, deletion, or modification, then the system has to ignore this transaction and do not re-execute it.

The "**Transaction Catalog Manager**" is added in order to have a history of transactions, which is complete (all details of transactions) and useful (i.e., easy-to-use by Side Effect Recovery Manager). For each transaction, it saves its commit time, the specified insert, delete, or update operation (since we suppose that each transaction include only one data manipulation operation), data read from the database, data written to the database, data values provided by the user in its data manipulation operation.

4.3 Implementation

The system prototype which supports our approach and shows its feasibility is yet under development (at the University of Sfax). It is being developed as a temporal stratum [23] on top of the existing XML DBMS xDB [24]. The goal is to maximally reuse the facilities of an existing XQuery Update Facility [18] implementation.

Currently, the prototype system allows only determining periods of inconsistencies when it receives a temporal XQuery Update Facility query with retroactive effect. The first author (Hind Hamrouni) is extending the prototype TempoXUF-Tool [16], developed within her master's project for temporal

XML data management under schema versioning, to support detecting and repairing data inconsistencies resulting from retroactive updates.

Fig. 12 reports the main TempoXUF-Tool GUI, which shows (i) a retroactive temporal XQuery Update Facility query specified by end user about changing a past salary of an employee, and (ii) the detected period of inconsistency with its sub-periods.

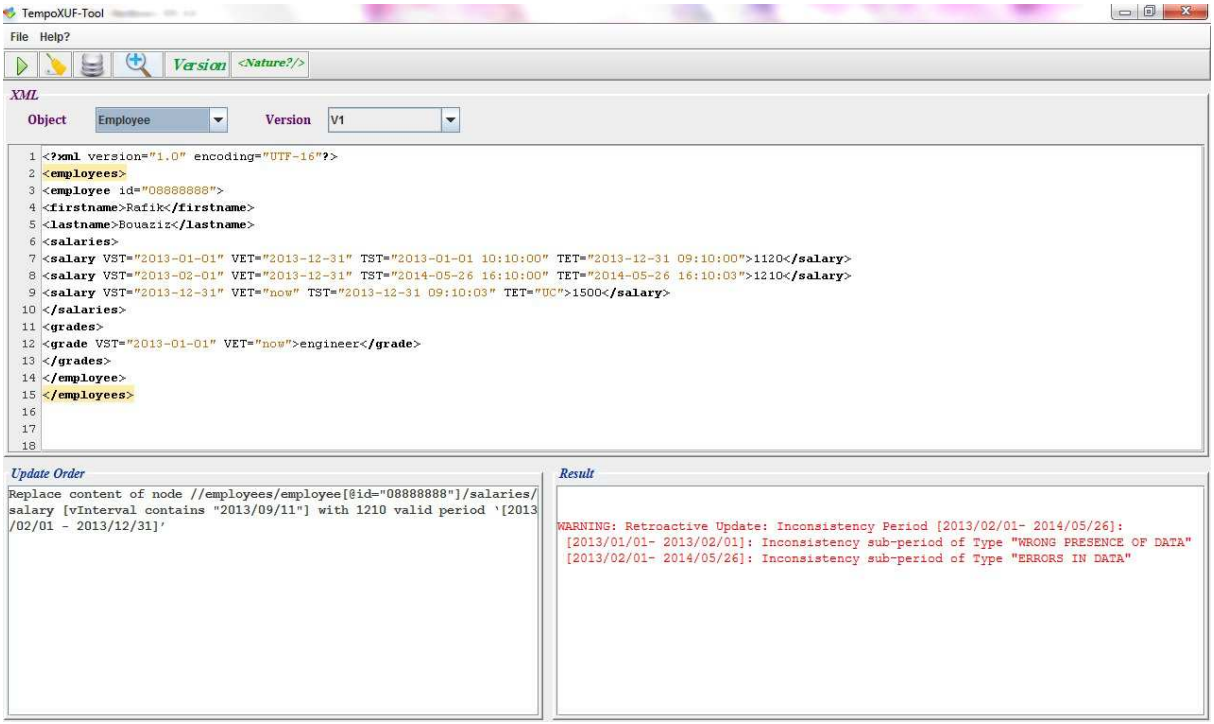

Fig. 12. Main GUI of TempoXUF-Tool.

5 Related Work

Despite the importance of preserving consistency of databases after retroactive updates, this issue has been considered only to a limited extent in current literature. However, it has not yet been studied in a multi-temporal and multi-version XML environment.

In the following, we discuss only the few works which are more strictly related with our proposal (i.e., [19], [25], and [26]), and we just briefly review some of the studied aspects which are somehow related with our work.

In [19, 25], the authors proposed a solution for redressing side effects generated by a retroactive update, named "correction propagation". This solution is defined to repair only inconsistencies which affect cumulative attributes (i.e., attributes which can undergo only operations of additions or subtractions of values, like the balance of a bank account or the turnover of a company). It is performed as follows: the correction of a past value "v" of a cumulative attribute must be propagated to all values that have been assigned to this attribute after "v". So, this solution does not preserve the database consistency under retroactive updates of values of attributes which are not cumulative. Furthermore, this solution was defined for a temporal relational environment which does not support schema versioning.

In [26], the author proposed the use of temporal active rules and retroactive rules [27] in order to redress side effects resulting from a retroactive update in a temporal relational active database. An active rule is an Event-Condition-Action (E-C-A) rule where: E is a basic or composite event; C is either (i) a boolean expression, or (ii) a query in the database query language that results in a true/false

answer; A is an execution of a database operation or an arbitrary application program. An active rule is said to be temporal if (1) the event is temporal, or (2) the condition is temporal. A retroactive rule is a rule whose action includes a retroactive update. The proposed solution was also limited since (i) it was devoted to a relational setting, and (ii) it used several triggers and heavy procedures that affect negatively the performance of the system.

Preserving the consistency of temporal databases was studied in other works which did not consider retroactive updates. Indeed, some of these works have dealt with database consistency with regard to (i) the concurrency control of transactions, by proposing new pessimistic [28, 29] and optimistic algorithms [30, 31], or (ii) the forensic analysis of database tampering [32, 33].

Retroactive and proactive updates were studied in temporal active databases [8] and in conventional (non-temporal) databases [34]. However, none of these works has dealt with inconsistencies that could result from such updates.

Campo et al. [35] studied the problem of validating a set of temporal constraints in a temporal XML document, by (i) presenting the main kinds of inconsistencies that may appear in a temporal XML document, and (ii) proposing methods for checking the presence of inconsistencies in a document, and fixing them. They also address documents where more than one consistency condition is violated (i.e., documents which include combined inconsistencies).

Martinez et al. [36] propose the concept of inconsistency management policies (IMPs) and show that IMPs (i) provide end users with tools that allow them to use the policies that they want, and (ii) allow removing either all tuples involved in the inconsistency or only a subset of them (i.e., a part of the inconsistency could persist in the database), and (iii) could be embedded as operators within the relational algebra.

In [37], the authors propose a generic methodology for the management of XML data update in XML-enabled databases. Such a methodology preserves the conceptual semantic constraints and avoids inconsistencies in XML data during update operations. But, the authors did not deal with retroactive updates, and define a data inconsistency as a data invalidity resulting from an XML data update, i.e., the updated XML data becomes invalid and not conforming to its original schema.

Brahmia et al. [15] and Hamrouni [16] have studied data management in multi-temporal XML databases supporting schema versioning, but none of them have taken into account data inconsistencies resulting from update operations performed on past data.

Svirec et al. [38] propose an algorithm to repair violations for a given XML document and a set of integrity constraints, specifically so-called functional dependencies [39]. Such an algorithm incorporates a weight model and a user interaction into the process of detection and subsequent application of appropriate repair of inconsistent XML documents.

Afrati et al. [40] have dealt with managing inconsistency in databases, within the framework of database repairs [41]; a repair of an inconsistent database is a database over the same schema that satisfies the integrity constraints at hand and differs from the given inconsistent database in some minimal way. More precisely, the authors study the problem of repair checking about inconsistent databases, which is the following decision problem: given two databases d1 and d2, is d2 a repair of d1? Basically, they propose two polynomial-time algorithms for repair checking.

Consistency of data, which takes into account the violation of semantic rules defined over a set of data items, has been also studied within the issue of data cleansing [42] and considered as a data quality dimension [43].

Zellag et al. [44] propose an approach for detecting consistency anomalies and automatically reducing their occurrence, in multi-tier architectures. Since the authors introduce a completely DBMS-independent approach, they propose that the system implementing their approach should be embedded into the middle-tier (which is represented usually by an application server, between a Web server and the database backend tier).

6 Conclusion

In this work, we focused on the problem of inconsistencies in multi-temporal XML databases: we proposed an approach for an automatic and graceful repairing of data consistency in such databases, resulting from retroactive updates. It allows the database management system (i) to detect any database inconsistency that happens after a retroactive update operation, and (ii) to perform necessary processings, in a transparent way, in order to repair inconsistencies automatically.

We think that our approach (i) maintains effectively the consistency of the database, and (ii) provides a low-impact solution since it requires neither modifications of existing temporal database schema (which include temporal integrity constraints), nor extensions to existing temporal XML models (e.g., τXSchema [45] and XBiT [46]) and query languages (e.g., τXQuery [47] and TXPath [48]).

As a part of our future work, we envisage to extend our work by (i) dealing with retroactive updates which concern several temporal XML elements (in our present work we have supposed that a retroactive update consider always one temporal XML element), and (ii) studying transactions that include several temporal XML updates with retroactive updates (in fact, in the current work we supposed that a transaction include always a single temporal XML update).

Furthermore, we also plan to study how to repair inconsistencies resulting from on-time and proactive updates of temporal XML databases, since the update of a current or a future temporal XML element could also give rise to some inconsistencies.

7 References

[1] Etzion O., Jajodia S., Sripada S., (eds.), *Temporal Databases: Research and Practice*, LNCS 1399, Springer-Verlag, 1998.

[2] Jensen C. S., Snodgrass R. T., "Temporal Database," in Liu L., Özsu M.T., (Eds.), *Encyclopedia of Database Systems*, Springer US, 2009, pp. 2957-2960.

[3] Grandi F., "Temporal Databases", in M. Koshrow-Pour, (Ed.), *Encyclopedia of Information Science and Technology (3rd Ed.)*, IGI Global, Hershey, (in press, to appear in July 2014).

[4] Jensen, C. S., Dyreson, C. E., (Eds.), Böhlen, M., Clifford, J., Elmasri, R. A., Gadia, S. K., Grandi, F., Hayes, P., Jajodia, S. K., Käfer, W., Kline, N., Lorentzos, N., Mitsoupoulos, Y., Montanari, A., Nonen, D., Peressi, E., Pernici, B., Roddick, J. F., Sarda, N. L., Scalas, M. R., Segev, A., Snodgrass, R. T., Soo, M. D., Tansel, A. U., Tiberio, P., Wiederhold, G., "The Consensus Glossary of Temporal Database Concepts – February 1998 Version", In O. Etzion, S. Jajodia, & S. Sripada, (Eds.), *Temporal Databases: Research and Practice*, LNCS 1399, pp. 367–405. Berlin, Germany: Springer-Verlag.

[5] De Castro C., Grandi F., Scalas M. R., "Schema versioning for multitemporal relational databases", Information Systems, 22 (5), 1997, pp. 249-290.

[6] Grandi F., Mandreoli F., "A formal model for temporal schema versioning in object-oriented databases", *Data and Knowledge Engineering*, 46 (2), 2003, pp. 123–167.

[7] Brahmia Z., Mkaouar M., Chakhar S., Bouaziz R., "Efficient management of schema versioning in multi-temporal databases," *International Arab Journal of Information Technology*, 9 (6), pp. 544–552, 2012.

[8] Etzion O., Gal A., Segev A., "Retroactive and Proactive Database Processing", Proceedings *of the 4th International Workshop on Research Issues in Data Engineering: Active Database Systems (RIDE-ADS 1994)*, Houston, Texas, USA, 14-15 February 1994, pp. 126-131.

[9] Bourret R., "XML and Databases", available at: <http://www.rpbourret.com/xml/XMLAndDatabases.htm>, last updated in September 2005.

[10] Dyreson C. E., Grandi F., "Temporal XML," in L. Liu and M. T. Özsu (Eds.), *Encyclopedia of Database Systems*. Heidelberg, Germany: Springer-Verlag, 2009, pp. 3032–3035.

[11] Brahmia Z., Grandi F., Oliboni B., Bouaziz R., "Schema Change Operations for Full Support of Schema Versioning in the tauXSchema Framework", *International Journal of Information Technology and Web Engineering* (in press, to appear during 2014).

[12] Zaniolo C., Wang F., "Temporal queries and version management in XML-based document archives", *Data and Knowledge Engineering, 65* (2), 2008, pp. 304-324.

[13] Clifford J., Croker A., Grandi F., Tuzhilin A., "On Temporal Grouping", *Proceedings of the International Workshop on Temporal Databases*, Zürich, Switzerland, 17-18 September 1995, pp. 194–213.

[14] Brahmia Z., Grandi F., Oliboni B., Bouaziz R., "High-level Operations for Changing Temporal Schema, Conventional Schema and Annotations, in the τXSchema Framework", TimeCenter Technical Report TR-96, 56 pages, January 2014. <http://timecenter.cs.aau.dk/TimeCenterPublications/TR-96.pdf>

[15] Brahmia Z., Bouaziz R., "Data Manipulation in Multi-Temporal XML Databases Supporting Schema Versioning", *Proceedings of the 4th International EDBT Workshop on Database Technologies for Handling XML Information on the Web (DaTaX'09)*, Saint-Petersburg, Russia, 22 March 2009.

[16] Hamrouni H., *Study of Data Management in Multi-temporal XML Databases Supporting Schema Versioning – Extending XQuery Update Facility to Temporal and Versioning Aspects*, Master thesis in Computer Science, Faculty of Economics and Management of Sfax, Tunisia, December 2012.

[17] Tatarinov I., Ives Z. G., Halevy A. Y., Weld D. S., "Updating XML", *Proceedings of the ACM SIGMOD Conference 2001*, Santa Barbara, California, USA, 2001, pp. 413-424.

[18] W3C, XQuery Update Facility 1.0, *W3C Candidate Recommendation*, 17 March 2011. <http://www.w3.org/TR/2011/REC-xquery-update-10-20110317/>

[19] Bouaziz R., *Temporal Management and Historisation of Data in Information Systems*, PhD thesis in Computer Science, Faculty of Science of Tunis, Tunisia, December 1991.

[20] Allen J. F., "Maintaining Knowledge About Temporal Intervals," *Communications of the ACM*, 26 (11), 1983, pp. 832-843.

[21] Skyt J., Jensen C. S., Mark L., "A foundation for vacuuming temporal databases", *Data and Knowledge Engineering*, 44 (1), 2003, pp. 1-29.

[22] Hellerstein J. M., Stonebraker M., Hamilton J., "Architecture of a Database System", *Foundations and Trends® in Databases*, 1 (2), 2007, pp. 141-259.

[23] Torp K., Jensen C. S., Snodgrass R. T., "Stratum Approaches to Temporal DBMS Implementation", *Proceedings of the 1998 International Database Engineering and Applications Symposium (IDEAS 1998)*, Cardiff, Wales, U.K., 8-10 July 1998, pp. 4-13.

[24] EMC, Documentum xDB, 2014. Available at: <http://www.emc.com/products/detail/software2/documentum-xdb.htm>

[25] Bouaziz R., Moalla M., "Historisation of Data and Recovery of Side Effects" (in french), *Proceedings of the 14th Journées de Bases de Données Avancées (BDA'1998)*, Hammamet, Tunisia, 23-26 October 1998, pp. 487-507.

[26] Samet A., *Automatic Recovery of Side Effects in a Multi-Version Environment*, Master Thesis in Computer Science, Faculty of Science of Tunis, Tunisia, March 1997.

[27] Pissinou N., Snodgrass R. T., Elmasri R., Mumick I. S., Özsu M. T., Pernici B., Segev A., Theodoulidis B., Dayal U., "Towards an Infrastructure for Temporal Databases: Report of an Invitational ARPA/NSF Workshop", *SIGMOD Record*, 23 (1), 1994, pp. 35-51.

[28] Finger M., McBrien P., "Concurrency Control for Perceivedly Instantaneous Transactions in Valid-Time Databases", *Proceedings of the 4th International Workshop on Temporal Representation and Reasoning (TIME 1997)*, Daytona Beach, Florida, USA, 10-11 May 1997, pp. 112-118.

[29] De Castro C., "On Concurrency Management in Temporal Relational Databases", *Proceedings of the 6th Italian Symposium on Advanced Database Systems (SEBD 1998)*, Ancona, Italy, 1998, pp. 189-202.

[30] Makni A., R. Bouaziz, "Performance evaluation of an optimistic concurrency control algorithm for temporal databases", *Proceedings of the 2nd International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2010)*, Menuires, France, 11-16 April 2010, pp. 75–81.

[31] Shirvani M. H., Mohsenzadeh M., Shirvani S. M. H., "A New Concurrency Control Algorithm in Temporal Databases", *Journal of Advances in Computer Research*, 4 (2), 2013, pp. 63-73.

[32] Snodgrass R. T., Yao S., Collberg C. S., "Tamper Detection in Audit Logs", *Proceedings of the 30th*

International Conference on Very Large Data Bases (VLDB 2004), Toronto, Canada, 31 August - 3 September 2004, pp. 504-515.

[33] Pavlou K. E., Snodgrass R. T., "Generalizing database forensics", *ACM Transactions on Database Systems*, 38 (2), 2013, paper 12.

[34] Deng M., Sistla A. P., and Wolfson O., "Temporal Conditions with Retroactive and Proactive Updates", *Proceedings of the 1st International Workshop on Active and Real-Time Database Systems (ARTDB-95)*, Skövde, Sweden, 9-11 June 1995, pp. 122-141.

[35] Campo M., Vaisman A. A., "Consistency of Temporal XML Documents", *Proceedings of the 4th International XML Database Symposium (XSym 2006)*, Seoul, Korea, 10-11 September 2006, pp. 31-45.

[36] Martinez M.V., Parisi F., Pugliese A., Simari G. I., Subrahmanian V., "Inconsistency management policies", *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*, Sydney, Australia, 16-19 September 2008, pp. 367–377.

[37] Pardede E., Rahayu J. W., Taniar D., "XML data update management in XML-enabled database", *Journal of Computer and System Sciences, 74* (2), 2008, pp. 170-195.

[38] Svirec M., Mlýnková I., "Efficient Detection of XML Integrity Constraints Violation", *Proceedings of the 4th International Conference on Networked Digital Technologies (NDT 2012) - Part I*, Dubai, UAE, 24-26 April 2012, pp. 259-273.

[39] Vincent M. W., Liu J., "Functional Dependencies for XML", Proceedings of the 5th Asian-Pacific Web Conference (APWeb 2003), Xian, China, 23-25 April 2003, pp. 22-34.

[40] Afrati F. N., Kolaitis P. G., "Repair Checking in Inconsistent Databases: Algorithms and Complexity", *Proceedings of the 12th International Conference on Database Theory (ICDT 2009)*, St. Petersburg, Russia, 23-25 March 2009, pp. 31-41.

[41] Arenas M., Bertossi L., Chomicki J., "Consistent Query Answers in Inconsistent Databases", *Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1999)*, Philadelphia, Pennsylvania, USA, 31 May – 2 June 1999, pp. 68-79, 1999.

[42] Mezzanzanica M., Boselli R., Cesarini M., Mercorio F., "Automatic Synthesis of Data Cleansing Activities", *Proceedings of the 2nd International Conference on Data Management Technologies and Applications (DATA 2013)*, Reykjavík, Iceland, 29-31 July 2013, pp. 138-149.

[43] Batini C., Scannapieco M., (eds.), *Data Quality: Concepts, Methodologies, and Techniques*, Springer, Heidelberg, 2006.

[44] Zellag K., Kemme B., "Consistency anomalies in multi-tier architectures: automatic detection and prevention", *The VLDB Journal, 23* (1), 2014, pp. 147-172.

[45] Snodgrass R. T., Dyreson C. E., Currim F., Currim S., Joshi S., "Validating Quicksand: Schema Versioning in τXSchema", *Data Knowledge and Engineering*, 65 (2), 2008, pp. 223-242.

[46] Wang F., Zaniolo C., "XBiT: An XML-based Bitemporal Data Model", Proceedings *of the 23rd International Conference on Conceptual Modeling (ER 2004)*, Shanghai, China, 8-12 November 2004, pp. 810-824.

[47] Gao D., Snodgrass R.T., "Temporal slicing in the evaluation of XML documents", *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB 2003)*, Berlin, Germany, 9-12 September 2003, pp. 632-643.

[48] Rizzolo F., Vaisman A. A., "Temporal XML: modeling, indexing, and query processing", *The VLDB Journal*, 17 (5), 2008, pp. 1179-1212.