Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2013) 21: 274 – 299

doi:10.3906/elk-1105-11

A temporal XML data model in anaesthesia information systems

Oya ÜNLÜ DUYGULU, Taflan İmre GÜNDEM*
Department of Computer Engineering, Boğaziçi University,
Bebek 34342, İstanbul, Turkey

Received: 06.05.2011 • **Accepted:** 24.10.2011 • **Published Online:** 27.12.2012 • **Printed:** 28.01.2013

Abstract: Time information is an important aspect of a health information system. Health information cannot be represented and queried correctly without time in medical research, auditing, and medicolegal cases. Extensible Markup Language (XML) is being used in health information systems and has been formerly proposed as a data model for anaesthesia information systems. In this paper, we propose incorporating time into an XML model for anaesthesia information systems. A new storage structure and a temporal index structure are proposed in order to store temporal XML data and process time queries efficiently. In our proposed model, time has 4 dimensions: valid time, transaction time, event time, and availability time. We discuss the advantages of attaching the time information to the data in anaesthesia records. We implemented and tested the proposed model and the storage structures.

Key words: Anaesthesia information systems, electronic medical records, time information, XML, multidimensional indices, AR* tree, Double R (2R) tree

1. Introduction

Anaesthesia documentation systems have been around for a long time. Computerised anaesthesia systems have become inevitable for various reasons such as accuracy and ease of use. The importance and advantages of computerised anaesthesia information systems over paper-based anaesthesia records have been stated by Parmar [1].

Gardner and Peachey [2] proposed an Extensible Markup Language (XML) schema for computerised anaesthesia records. In [2], it was claimed that in the near future, a nationally or internationally standard XML schema will be adapted to anaesthesia records. Additionally, a Health Level Seven (HL7) Special Interest Group (SIGGAS) was formed to create HL7 clinical document architecture (CDA)-compliant schemas from preexisting anaesthetic XML schema resources [3].

Health systems such as electronic medical records, hospital information systems, decision support systems, and anaesthesia information systems contain a significant amount of time-related data [4]. Storing time-related anaesthesia information is useful in academic research, anaesthesia audits, and medicolegal courts. Recording time information helps with detecting illegal modifications and malpractices in anaesthesia audits [5].

Time-related anaesthesia information helps to answer some legal questions in critical situations. For example, a system that stores time-related data will be able to accurately answer the question: "When the doctor prescribed penicillin on 14 August 2007, did the doctor know or did the software system contain the information (at the time of prescribing) that the patient had an allergic reaction to penicillin?" Such a question

*Correspondence: gundem@boun.edu.tr

may arise in the case of a doctor being accused of malpractice. Moreover, illegal modifications to the records can be easily detected if a specific type of time information (transaction time, to be explained later) is attached to the data.

In this paper, combining the requirements of XML and time, we propose a temporal XML (TXML) data model for anaesthesia data with 4 different temporal dimensions: valid time, transaction time, event time, and availability time. We also propose efficient physical storage structures to support the model. The proposed data model supports the time information and the specially designed data structures provide efficient query evaluation for various types of temporal queries. The proposed system and 2 different alternative systems are implemented and compared for performance.

In the next section, we provide preliminary information. In Section 3, we present our model. In Section 4, we provide the performance study of the proposed system. In the last section, we present our conclusions.

2. Background

2.1. Temporal dimensions

There are several kinds of temporal dimensions defined in the literature [6]: valid time, transaction time, event time, and availability time. The following imaginary scenario will be used in explaining the meaning of the 4 time dimensions.

On 10 June 2007, at 1300 hours, before the intraoperative period, the anaesthetist prescribed an ephedrine-based therapy to be applied from 1500 to 1900 hours, during the surgery. Therapy records were entered into the anaesthesia information system at 1400 hours. However, a change in the preoperative laboratory results occurred at 1555 hours. The new preoperative laboratory results were entered into the system at 1600 hours by the laboratory assistant. The anaesthetist had been notified of this change at 1600 hours. Consequently, the anaesthetist decided to make a change in the patient's therapy. The ephedrine infusion was stopped at 1615 hours and replaced by a diazepam-based therapy, to be applied from 1625 to 1900 hours. The new facts are entered into the anaesthesia system at 1700 hours.

2.1.1. Valid time

"The valid time of a fact is the time when the fact is true in the modelled reality" [6]. Valid time information helps identify the validity period of anaesthesia data. The validity of a fact cannot be expressed by a single timestamp. The valid time includes the start time and the end time of the validity period and is denoted by an interval V = [Vs, Ve]. The end point of the valid time interval of a fact is sometimes unknown during data insertion, i.e. the end point follows the current time. Current time is denoted by "Now". When a fact's validity ends in the real world, the end point of the validity interval is set to the end date in the database. Temporal databases can support current time in a number of different ways [8]. Maximum approach represents the current time as unrealistically large dates such as '31-December-9999'.

In our scenario, the anaesthetist prescribes an ephedrine-based therapy to be applied between 1500 and 1900 hours. This drug information, ephedrine therapy, has to be stored in the information system with the validity period of 1500–1900 hours. With the help of valid time, the question "When was the ephedrine therapy applied to the patient?" can be answered.

2.1.2. Transaction time

Transaction time is used for recording all actions, including insertion, deletion, and update operations in the information system. "The transaction time of a fact is the time when the fact is current in the database

system and may be retrieved" [6]. As a consequence, transaction times are generally not time instants but are durations. When an object is stored in the database, the end point of its transaction time interval is set to "until changed" (UC), and then when the object is deleted, the system sets the transaction time end point to the deletion date. Hence, the deletion is purely logical and the object is not physically removed but ceases to be part of the database's current state.

In the preceding scenario, the drug information, ephedrine therapy with the validity period of 1500–1900 hours, is inserted into the system at 1400 hours. Thus, its transaction time period is 1400-UC. UC stands for "until change" and means that the data are current in the system. If the transaction time associated with a data item is available, the data item is never physically deleted, but instead "ceases to represent the systems' current state" [7]. At 1700 hours, when the new facts associated with the ephedrine therapy are entered into the anaesthesia system, the UC value of ephedrine therapy (i.e. the transaction time end point) becomes 1700 hours, but the ephedrine therapy record is not deleted from the database system. Thus, the question "When did the anaesthetist change the ephedrine-based therapy in the information system?" can be answered with the transaction time information. In the above scenario, if the information system supports only the transaction time (but not the valid time), then there will be 1 record for the ephedrine therapy and 1 for the diazepam therapy. Since the validity intervals of the drugs are not recorded, the question "When was the diazepam or ephedrine therapy applied to the patient?" will not be answered. With the transaction time, one can only know that ephedrine and diazepam therapies are applied to the patient; one cannot know when. Although the anaesthetist changes his/her therapy decision during the intraoperative period, a system that supports only the transaction time cannot show this change. On the other hand, if a system supports only valid time, then again there will be 1 record for the ephedrine therapy and 1 for the diazepam therapy. Until 1700 hours, the ephedrine therapy record shows that the therapy is to be applied from 1500 to 1900 hours. However, because of the therapy change, the validity period of this record is not true after 1615 hours. Then, at 1700 hours, the validity period of the ephedrine therapy is modified as 1500–1615 hours. If the current valid drug therapy is queried at 1630 hours, the system will return ephedrine (since the new therapy information is recorded at 1700 hours), even though the correct answer is diazepam. Moreover, one cannot know when the records are entered into the information system.

For this reason, in order to retrieve accurate results for specific queries, both the valid time and the transaction time of anaesthesia data have to be stored in the system. In a system that supports the valid and transaction times, there will be 2 records for the ephedrine and 1 record for the diazepam therapy in our example. Until 1700 hours, there is only 1 record for the ephedrine therapy, with the validity period of 1500–1900 hours and the transaction time period of 1400-UC. However, at 1700 hours, the new therapy decision is inserted into the system, and the first record is no longer valid and should be deleted. However, in a system that supports the transaction time, no record is deleted from the system, but instead its transaction time is updated. Because of this, the first record's transaction time period is set to 1400–1700 hours. This means that the record was current in the system until 1700 hours. At the same time, a new record for the ephedrine therapy is inserted into the system. The new ephedrine record's validity period is set to 1400–1615 hours and its transaction time period is set to 1700-UC (UC shows that this record is current). In this system, a question that asks "What were the prescriptions of the anaesthetist?" will return "ephedrine therapy 1500–1900, ephedrine therapy 1500–1615, and diazepam therapy 1625–1900" prescriptions, whereas a valid time system will return only "ephedrine therapy 1500–1615 and diazepam therapy 1625–1900" prescriptions. Moreover, a transaction time system would return ephedrine and diazepam therapy without their validity periods.

Although valid and transaction times are the most commonly used temporal dimensions in the literature, they are insufficient in distinguishing retroactive updates from delayed updates. Event and availability times are also needed in order to be able to analyse anaesthesia information under some circumstances.

2.1.3. Event time

Valid and transaction times are insufficient in distinguishing retroactive and delayed updates. "Event time of a fact is the occurrence time of a real-world event that either initiates or terminates the validity interval of the fact" [9]. A fact can be initiated by an event and can be terminated by another event. Consequently, the end point of the event time of a fact can be null if the fact is not terminated by an event or can be a point in time if the fact is terminated.

In our scenario, the anaesthetist decided to apply the ephedrine therapy at 1300 hours. The initiating event of the ephedrine therapy is the decision of the anaesthetist. Attaching a valid time and a transaction time to the therapy data causes 2 records for the ephedrine therapy. In the first ephedrine record, the start point of the event time (i.e. the initiating event time) is set to 1300 hours. At that time, since there is no terminating event for the ephedrine therapy, the end point of the event time (i.e. terminating event time) is also set to 1300 hours. However in our scenario, the ephedrine infusion is stopped and the therapy is continued with diazepam because of the change in the preoperative laboratory results. The terminating event of the ephedrine therapy then becomes the therapy change decision of the anaesthetist. The anaesthetist decided to continue with a diazepam-based therapy instead of the ephedrine therapy at 1600 hours. Thus, in the second ephedrine record, which contains the current validity interval, the event time start point is set to 1600 hours. Since there is no terminating event for this therapy, the event time end point is also set to 1600 hours. In a system that supports the event time, a question that asks "When did the anaesthetist change his/her decision about the ephedrine therapy?" will return 1600 hours, whereas in a valid or transaction time system, this question cannot be answered.

However, using only the event, valid, and transaction times, it is impossible to express the time when the doctor became aware of the change in the laboratory results.

2.1.4. Availability time

The last time dimension is the "availability time, which is the time interval during which the fact is known and believed correct by the information system" [10]. It is the time when someone or something associated with the information system becomes aware of a fact. The end point of the interval is the time at which the information system realises that the fact is not correct. As for the transaction time, the end point UC means that the fact is currently believed correct. In the anaesthesia documents, information may not be entered into the system as soon as it appears. Because of this, the time that an information system (the anaesthesia system or the anaesthetists) becomes aware of a fact does not always coincide with the transaction time or the event time of a fact.

In our scenario, the question "Why did the anaesthetist apply the ephedrine-based therapy between 1555 and 1600 hours, although the preoperative laboratory results had been corrected at 1555 hours?" can only be answered by the availability time.

According to the scenario, the availability time period of the new preoperative laboratory results is 1555-UC. The end point is UC because the data on laboratory results are currently available. In addition to this, the transaction time period of the new preoperative laboratory results is 1600-UC. With this information, it is obvious that the new results were entered into the system 5 min later than the time at which they had been

available. Thus, the anaesthetist had been notified of the new results at 1600 hours. With the help of the availability and transaction time, the anaesthetist cannot be accused of the therapy between 1555 and 1600 hours.

Figure 1 represents a part (the drugs applied to the patient) of our scenario as an anaesthesia record in XML format using time dimensions. The 4 different time dimensions in XML format are present in it. In Figure 1, the time information of the data is contained in the elements of TimeElement type. The elements VT, TT, ET, and AT stand for valid time, transaction time, event time, and availability time, respectively. The start point of a time interval is represented by a Low attribute and the end point is represented by a High attribute.

```
<Drugs>
    <Drug>
        <DrugName>Ephedrine</DrugName>
        <TimeElement>
            <VT Low="10.06.2007 15:00" High="10.06.2007 19:00"/>
            <TT Low ="10.06.2007 14:00" High ="10.06.2007 17:00"/>
            <ET Low ="10.06.2007 13:00" High ="10.06.2007 13:00"/>
            <AT Low ="10.06.2007 13:00" High ="10.06.2007 16:00"/>
        </TimeElement>
        <TimeElement>
            <VT Low="10.06.2007 15:00" High="10.06.2007 16:15"/>
            <TT Low ="10.06.2007 17:00" High ="UC"/>
            <ET Low ="10.06.2007 13:00" High ="10.06.2007 16:00"/>
            <AT Low ="10.06.2007 16:00" High ="UC"/>
        </TimeElement>
    </Drug>
    <Drug>
        <DrugName>Diazepam</DrugName>
        <TimeElement>
            <VT Low="10.06.2007 16:25" High="10.06.2007 19:00"/>
            <TT Low ="10.06.2007 17:00" High ="UC"/>
            <ET Low ="10.06.2007 16:00" High ="10.06.2007 16:00"/>
            <AT Low ="10.06.2007 16:00" High ="UC"/>
        </TimeElement>
    </Drug>
</Drugs>
```

Figure 1. XML representation of the drugs applied and their time dimensions in the given anaesthesia scenario.

The ephedrine therapy in the scenario is related to the first 2 drug records in Figure 1. The first record of the ephedrine is current until 1700 hours. After 1700 hours, the second ephedrine record becomes current. The diazepam therapy record is related to the third record.

In the literature, there is some research on creating a common XML schema for anaesthesia documents [2,10]. However, none of them consider the time information of the data. HL7 provides time data types for representing temporal data. It also supports timestamps, time intervals, and periodic time intervals as time data types. With the existing technologies, time information can be attached to anaesthesia data as a user-defined data type. For example, the start and end times of the surgery can be defined in the surgery information part of the anaesthesia document. However, using the user-defined temporal attributes has 2 disadvantages: 1) the users have to define all of the temporal attributes beforehand, and 2) the users are responsible for maintaining the time relations in the anaesthesia data model. In our proposed system, all of the time elements are present in the schema and do not have to be defined by the user. Additionally, queries on the time information are processed automatically by the system.

2.2. Index structures for multidimensional temporal data

Conventional databases capture the current information. They cannot reflect the previous information because update operations overwrite the previous data. Storing only the current data will result in the loss of information

as times goes by. What is needed is a database that efficiently supports the storing and querying of time-varying information.

Temporal database systems support efficient storage and retrieval of time-related data with the help of data structures that are specially designed for time. Since both temporal and spatial data have multiple dimensions, spatial indices can be adapted to indexing temporal data. Multidimensional index structures used for spatial data indexing are also good candidates for indexing temporal dimensions. In the literature, index structures such as R tree, R* tree, Double R(2R) tree, Multiversion B tree, Bitemporal R tree, Bitemporal Interval tree, and Adaptive R* tree are proposed for multidimensional data.

R* tree is one of the basic index structures that is frequently used in multidimensional indices. Although the R* tree can index time dimensions, it has the major disadvantages of overlapping and dead space that result from the maximum timestamp approach. When many intervals end at *now* (i.e. for the transaction time when the data are alive), keeping the now-relative data in a separate structure is a better solution.

Double-tree methodology [11] avoids the problem of overlapping while retaining the advantage of using off-the-shelf access methods. Although it is implemented by 2 R* trees, various other multidimensional access methods could be facilitated. When an object with valid-time interval I is inserted into the database at transaction-time t, it is inserted at the front R tree. The front R tree keeps the live objects for which the transaction endpoint is unknown. If a bitemporal object is later "deleted" at some transaction time t_e (t_e >t), it is physically deleted from the front R tree and inserted into the back R tree. The back R tree keeps the logically deleted objects with known transaction-time intervals.

As mentioned in [12], the multidimensional index structures, such as R* tree [13], X tree, and Kd tree [14], are designed for all-dimensional range queries in which a query range is given for each dimension. If the mentioned indices are used for partially dimensional (PD) range queries, then the information related to the irrelevant dimensions has to be accessed from the disk, also.

The AR* tree [12] is proposed for evaluating partially dimensional range queries efficiently. N-dimensional indices are often used for n-dimensional queries. However, queries do not always contain all of the dimensions. Although the index is built on an n-dimensional space, the range queries may use only d of the n dimensions (where d is smaller than n). The key concept behind an AR* tree [11] is to divide each of the n-dimensional R* tree nodes into n 1-dimensional nodes. Each node of an R* tree holds the information in all of the index dimensions. As stated in [11], the Adaptive R* tree has a clearly better performance for PD range queries than the naive methods, R* tree and Multi-B tree.

2.3. Related work in the literature

2.3.1. Anaesthesia documents

There exist various anaesthesia information systems (AIMSs). In [15], it was argued that AIMSs improve standardising care in surgery and facilitate computerised decision support and systematic decision support.

In [2], the need for an international standardised XML schema for computerised anaesthetic records was emphasised. The requirements for such an XML anaesthesia schema were proposed.

In [16], the need for an international anaesthetic XML-standard and its adoption to AIMSs was also emphasised. There was also a proposal for a postoperative report using Extensible Stylesheet Language Transformation (XSLT).

The important work that is being carried out by the HL7 SIGGAS includes the following, as reported in [3]:

- Identifying standards specific to anaesthesiology necessary for standardised quality and outcomes reporting, and measuring outcomes.
- Identifying required terminology for reporting and measurements.
- Identify requirements for standardised anaesthesia records to facilitate the exchange and aggregation of perioperative data.
- Identifying anaesthesia constraints against existing HL7 artefacts.

The work done in [4] was not specific to anaesthesia but was intended for general medical documents. It emphasises the importance of time in medical information systems and proposes an architecture called the temporal mediator to integrate temporal reasoning and temporal maintenance. For the temporal maintenance bitemporal model (also mentioned in the proceeding section), the structured query language (SQL) is utilised.

2.3.2. Time in XML

The important time-related work in the literature may be summarised as follows. Bitemporal data model XBIT [17] basically shows that the valid time, transaction time, and bitemporal databases can be naturally viewed in XML using temporally grouped data models that are compatible with the hierarchical structure of the XML. Although the XBIT data model is general and can be applied to historical representations of relational data and XML documents in native XML databases, it is a logical data model and no physical storage structures or any indexing structures that support efficient temporal management exist for the model. Moreover, XBIT supports only the valid-time and transaction-time dimensions.

In [18], a temporal XML data model for normative documents is proposed. The model is based on a hierarchical organisation of normative texts. There are 4 different temporal dimensions (valid time, transaction time, publication time, and efficacy time) in the model in order to represent the evolution of norms in time and their resulting versioning correctly. The model represents the norms in an XML-based data model, which is also enriched with timestamping to make versioning possible. The model is implemented on an XML-enabled system. Although the model in [18] supports multiple temporal dimensions, there are no special storage structures or indices for the temporal dimensions. Moreover, the temporal dimensions of "publication" and "efficacy" time are related to norms and do not have a general use.

In [19], a temporal XML data model is proposed for tracking the historical information in an XML document and for recovering the state of the document at any given time. Valid time is supported by the model, but the authors claim that the transaction time can also be applied in the model as the valid time. Although the model supports efficient implementations of temporal data models, only the valid time and transaction time dimensions are mentioned in the paper.

3. Proposed system

The system that we propose has 3 main objectives. These can be summarised as: 1) attaching the time information to the anaesthesia data, 2) efficiently storing the mentioned 4 different time dimensions for anaesthesia documents, and 3) efficiently processing the partially dimensional temporal queries.

In our system, we have different components for supporting the temporal data and indexing the XML documents. Figure 2 shows the overall architecture of our proposed system.

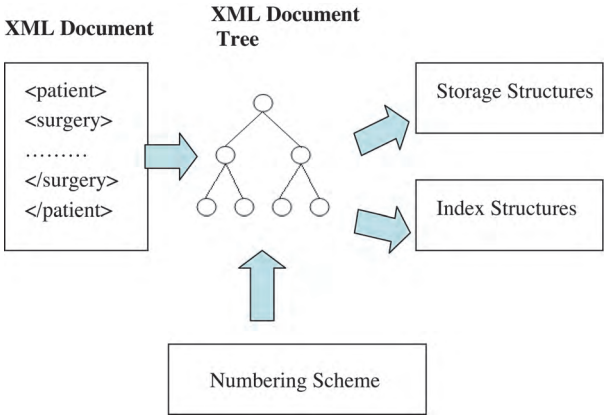

Figure 2. Overall architecture of the proposed system.

We propose a logical data model (TXML) to support the 4 time dimensions in the XML anaesthesia documents. In the logical data model, we use XPath without any modifications. In our system, we utilise a numbering scheme for labelling the document trees. To store the anaesthesia documents in TXML, we have 2 basic physical structures: storage and index structures. In the storage part, we extend the well-known storage structures for native XML database systems to support the time dimensions defined in our logical model.

To expedite the query evaluation, we have 3 different index structures in our proposed system. The basic idea behind our proposed index structures is similar to that in [20], which is to take advantage of the indexing paths rather than the nodes. We also have an index structure (TAR* tree) that supports both the PD range queries and the now-relative temporal dimensions.

In our proposed system, when a new temporal XML document is inserted, a tree is constructed and labelled using a specific numbering scheme. After the construction of the temporal XML tree, the other storage structures are created. The temporal and path indices are also created while the nodes of the tree are created. During a query evaluation, the mentioned storage structures and indices are accessed to expedite the retrieval of the temporal anaesthesia data that are stored using our system.

3.1. Logical data model

In the literature, temporal XML data models use 2 different approaches to attach the time information. The first is to attach the time information to the edges and the second is to define the time elements as subelements. In order to implement the former, the XPath model has to be modified. In our proposed temporal XML data model, TXML, we define a time element as consisting of 4 different time dimensions as subelements. Each of the subelements contains "low" and "high" attributes to represent the start and end points of its time interval. Updates on the time information of the elements cause multiple time elements to be created in an element. If one of the time values of an element is updated, the record is deleted and inserted into the database again. Although the element is the same, there are 2 timestamp elements associated with it. One is logically deleted and the other is alive.

Figure 3 shows the time elements in a temporal XML document and Figure 4 shows a tree representation of the document in Figure 3.

3.2. The numbering scheme used

A unique label (which shows the relationship between any 2 nodes and eliminates the need to access the actual documents) is assigned to each node in the tree representation of an XML document. As mentioned in [21], several node labelling techniques have been proposed in the literature [22,23]. A numbering scheme that requires relabelling will be inefficient for temporal XML documents because of the updates on the time elements. A new labelling scheme for dynamic XML data (LSDX) is defined in [21]. The LSDX supports the representation of ancestor, descendant, and sibling relationships between the nodes. Moreover, it provides updating of the XML data without having to modify the existing labels. Since it eliminates the need for relabelling, we used the LSDX in our system.

```
<Element id=1>
        <TimeElement id=1>
                <VT Low=10.00 High=13.00></VT>
                <TT Low=11.00 High=12.00></TT>
                <AT Low=10.00 High=12.00></AT>
                <ET Low=09.00 High=12.00></ET>
        </TimeElement>
        <TimeElement id=2>
                <VT Low=10.00 High=12.00></VT>
                <TT Low=12.00 High=UC></TT>
                <AT Low=12.00 High=UC></AT>
                <ET Low=12.00 High=12.00></ET>
        </TimeElement>
</Element>
```

Figure 3. Time dimensions of an element.

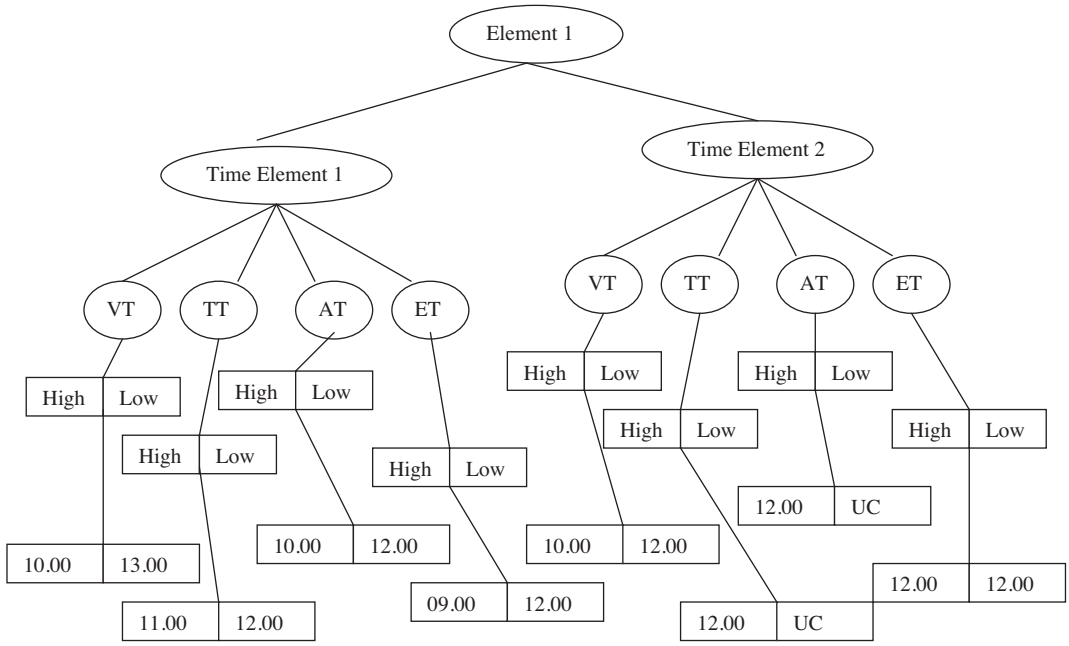

Figure 4. Tree representation of the document in Figure 3.

The LSDX uses both numbers and letters for labels. The root has level 0 and the next level is incremented by 1. Given a node with n child nodes u_1, $u_2 \ldots u_n$, the label for u_1 is a combination in the order of its level

plus its parent's label plus "." plus "b". The label for u_2 is the same as that for u_1, except that the last symbol is "c" and not "b". The next one has the last symbol in alphabetical order, which is "d". For example, the root node with 3 children has the label 0a. The first child of the root has the label 1a.b. The next child has the label 1a.c, and the last one has the label 1a.d. The first child of the node labelled 1a.b has the label 2ab.b. If there is no node before the place where a new node, u_i, will be inserted, the label of u_i will be the label of the node standing after it (assume node u_j) with "a" inserted after ".". Let us assume that u_j has the label 2ac.b., and then u_i will have the label 2ac.ab. For details, please refer to [21].

3.3. Storage structures: node types

In native XML database implementations [24–28], we usually have path indices and storage structures for the node types (nodes of an XML tree). The node types are basically categorised as internal nodes (elements) and external nodes (values). The TXML consists of 3 different types of nodes: internal nodes (elements), external nodes (contents), and timestamp nodes (time elements).

The structure of the internal and external nodes in the TXML is an extended version of the node types in the efficient native XML storage system [28]. Since the time dimensions are represented with a timestamp element in the TXML, a new node type, which efficiently stores the time dimensions, is proposed in our system.

3.3.1. Internal nodes

All of the nodes, except for the timestamp and the value nodes in the XML document tree, are stored in the internal node format (Figure 5). A unique document identifier is assigned to each document and a node identifier is assigned to each node, except for the external node, so that a node can be identified by the pair (document identifier, node identifier). The PathInfo field gives the path from the root to the node in question. The path info is computed by concatenating the node identifiers of the ancestor nodes, starting from the document id and its root to the parent of the current node. A node identifier is unique and is assigned to a newly inserted node according to the numbering scheme (Section 3.2). The Element Type field indicates the element type of the node. The Value Pointer points to the external node of the node, if the node has content. If not, the Value Pointer is null. The Timestamp Block Pointer field is a pointer to a timestamp block since the number of timestamp elements of a node can be more than one. The Timestamp Block Pointer of a node points to the live element in a block that consists of all of the timestamp elements of that node. The Child Block Pointer field is similar to the Timestamp Block Pointer in the sense that both fields point to a block of elements. The types of the elements in the Child Block are the same as those of the node that is currently being explained, since both are internal nodes. Both of the blocks contain node identifiers and a pointer to address tables, which are used to access the memory locations of the nodes.

3.3.2. External nodes

The external nodes (Figure 6) are designed to store the content of the internal nodes; they are actually equivalent to the value nodes in the XML document tree. The Document Identifier field identifies the document of the value node. The PathInfo field is calculated by concatenating the node identifiers of the ancestor nodes, starting from the document id and its root to the parent of the value node. The Parent Pointer points to the parent of the value node in the address table, which shows the memory location of the parent node. The Value field contains the content of the value node.

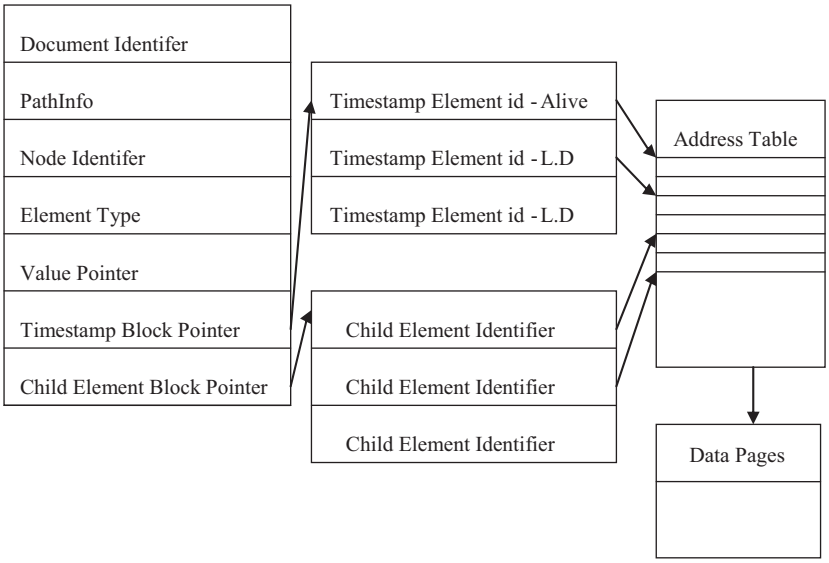

Figure 5. Structure of an internal node.

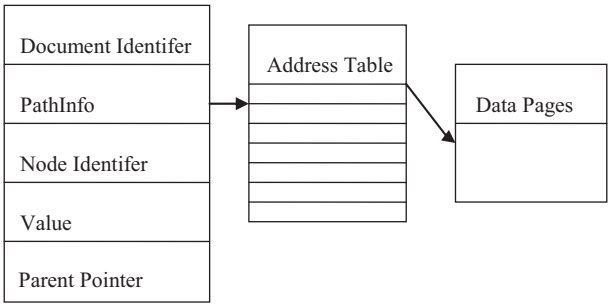

Figure 6. Structure of an external node.

3.3.3. Timestamp nodes

Timestamp node types (Figure 7) are proposed to store the timestamp elements efficiently. We have 4 different temporal dimensions in the TXML. These time dimensions are stored in the timestamp element as subelements. In order to make the subelements HL7-compatible, the high and low attributes are defined in each of them. The start point of the time dimensions is represented by the low attribute and the end point of the time dimension is represented by the high attribute. The Document Identifier field identifies the document associated with the timestamp node. The PathInfo field is calculated by concatenating the node identifiers of the ancestor nodes, starting from the document id and its root to the parent of the timestamp node. The Status field represents the state of the timestamp. The timestamp can be alive or logically deleted. The pointer to the next timestamp field points to the next timestamp of its parent node since the internal nodes may have more than one timestamp node.

3.4. Index structures

In query evaluations, we need special index structures to evaluate the queries without accessing the original documents. There are 2 types of indices that are proposed in our system: time indices and path indices.

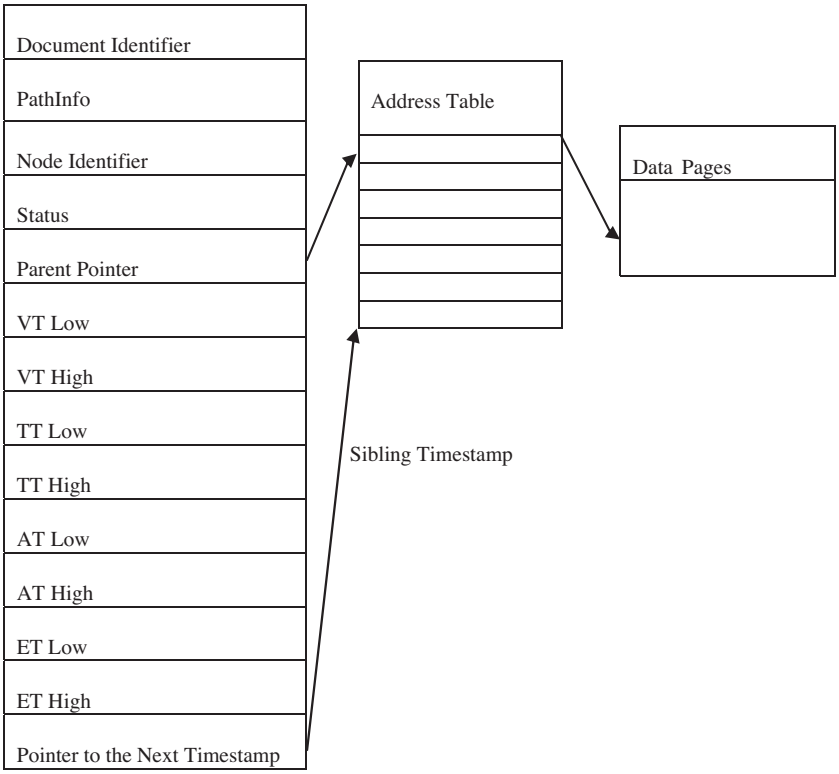

Figure 7. Structure of a timestamp node.

We propose time indices to process the time-related queries efficiently. The Temporal AR* tree (TAR* tree) that we propose combines the AR* tree (PD range query property) and the 2R tree (now-relative bitemporal data property) proposed in the literature and is used for supporting now-relative PD range queries.

Path indices in our system are used for supporting the selection and join operations that are used in the temporal query evaluations.

3.4.1. Path indices

Path index table

As mentioned in [20], indexing paths (that are valid during a certain interval) rather than nodes enhance the query performance dramatically. This ability is not provided by traditional path indices. The basic idea of our path index tables is similar to that in [20] (i.e. we take advantage of indexing paths rather than nodes). In [20], the authors index all equivalence classes of continuous paths in the documents in a separate table according to their path type. "Continuous path" means the paths that are valid in a certain interval. However, indexing all of the path types in a document will require a huge number of index tables. In our proposed system, instead of indexing all of the paths, we index all of the root-to-leaf paths and create an alternate solution for the nonleaf paths.

In our system, we define one path index table for each root-to-leaf path type. When a leaf element is inserted into the database, the path index table for that path type is also updated. For example, when a surgery element is inserted under a patient element, the corresponding path index table "AnaesthesiaDb/Patient/Surgery/Name" has to be updated. Figure 8 displays the structure of the path index table. The NodeId field represents the unique identifier of the inserted element. The PathInfo field is calculated by

concatenating the node identifiers of the ancestor nodes, starting from the document id and its root to the parent of the current node. The Status field indicates the status of the node, i.e. the node is current or logically deleted. The Timestamps field consists of the timestamp elements of the node. Timestamp elements can be more than one so that the field stores the node identifiers of the timestamp elements. When a new leaf element is inserted, it is also inserted into the corresponding time index. Each data object that is inserted into the time index has a minimum bounding rectangle (MBR) enclosing it. The MBR id of the element is inserted into the path table. The MBR field indicates the element's MBR and will be used in the temporal query processing.

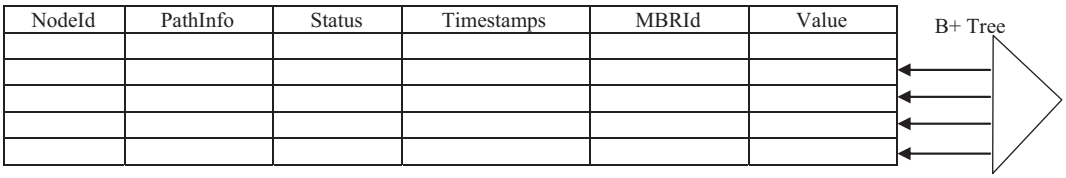

Figure 8. Structure of the path index table.

The value fields of all of the path index tables are connected to a B+ tree, which is used for range queries on the value fields. When a query that has a constraint on the value field is asked, the B+ tree is used for searching the value fields.

The Status field, the MBR List, and the B+ tree index are the distinguishing features of the proposed structure.

Join index table

The path index tables are used to index only the root to leaf paths. Nonleaf paths are not indexed with path index tables in order to not increase the number of index tables dramatically. When we analyse the queries, we realise that the nonleaf nodes are generally used for joining leaf nodes. Thus, we propose the join index table, which indexes all of the nonleaf nodes in one table. Figure 9 shows the structure of the join index table.

Element Type	Path Info	Node Id	Status	Time stamps	Path Index Instance	MBR List

Figure 9. Structure of the join index table.

When a node is inserted into the temporal XML document tree, if the node is a leaf node, the corresponding path index table is updated; otherwise, the join index table is updated.

The meaning and functions of the fields of the join index table are explained as follows. The Element type field is used to store the element type of the node because different element types are indexed in the same table. The NodeId is the unique identifier of the node. The PathInfo field is computed by concatenating the node identifiers of the ancestor nodes, starting from the document id and its root to the parent of the current node. The Status field indicates the status of the node, i.e. whether the node is current or logically deleted. The Timestamps field consists of the timestamp elements of the node. Timestamp elements can be more than

one so that the field stores the node identifiers of the timestamp elements. A nonleaf node may have one or more leaf children nodes or descendant nodes. A join operation on a leaf node uses an ancestor nonleaf node of the leaf node. The second leaf node in the join operation is a leaf node that has the same ancestor node in the upper levels of the tree. The Path Index Instances field contains the descendant leaf element records of the join element in the path index tables. If the element type is used in the path of an index table, it means that the nonleaf entry descendants are stored in that path index table. A nonleaf node in the join index table stores all of the children/descendant instances located in the path index table according to their path type. A hash map is used for path index instance storage. The keys are composed of the root-to-leaf path types and the values are the (PathInfo, NodeId) tuple of the nodes in the path tables, which are the children/descendants of the node in the join index table. The MBR List field is similar to the path index fields. When a leaf node, which is the descendant/child of a node in the join index table, is inserted, its corresponding path index table field is updated. Each insertion to the path table requires an update in the join index table for the parent or ancestor node. A node's timestamp must be greater than the union of its descendant/children timestamps, so the MBR of a child node must overlap with its parent/ancestor's MBR. The MBR List field of a node in the join index table stores all of the MBRs of its descendant leaf nodes.

3.4.2. Time indices

Time-related data have to be stored in special data structures so that they can be efficiently queried. There are some multidimensional indices that are defined in the literature [11,13,29]; however, none of them are designed to support the mentioned 4 time dimensions. Time dimensions have different properties from spatial dimensions because of their now-relative property. In order to use multidimensional indices for multiple time dimensions, modifications are needed. Furthermore, as mentioned in [12], most of the existing multidimensional indices are designed for all of the dimensional queries. However, temporal queries do not always use all of the time dimensions. Instead, they use different combinations of the 4 time dimensions. In the literature, the AR* tree [12] is designed to efficiently support partially dimensional range queries, but it does not support now-relative data.

The temporal characteristics of anaesthesia data define the requirements of a temporal index. Anaesthesia data have a validity period between the preoperative and postoperative anaesthesia periods, i.e. the valid time end interval is closed. Generally, when we insert data into the database and sometime later realise that it is incorrect, we delete the data. However, anaesthesia data must not be deleted for medicolegal reasons. If anaesthesia data are current in the database, then the end point of their transaction time dimension is equal to "now". The availability time has similar characteristics to the transaction time, i.e. if the data are current, then the availability end point is also equal to "now".

For these reasons, an index structure that supports both PD range queries and now-relative temporal dimensions is needed for the temporal indexing of anaesthesia data. In this paper, we propose a time index structure, the Temporal AR* tree (TAR* tree), which we designed for now-relative PD temporal range queries.

Temporal AR* tree

The TAR* tree combines the PD property of the AR* tree and the now-relative property of the 2R tree. The 2R tree is chosen as the multidimensional indexing method because it supports now-relative data for the transaction time dimension. Anaesthesia data are not now-related in terms of the valid time dimension. There is a proposed index structure that supports now-relative data for both the valid and transaction time [30]. However, one of the disadvantages of the structure in [30] is that it does not use off-the-shelf methods, and it is

very complex. Since anaesthesia data are valid between the preoperative and postoperative periods, the valid time end interval is closed. Because of this, a multidimensional temporal index structure, which supports the now-relative transaction time, is suitable for anaesthesia data. A temporal data model that supports the 4 time dimensions, where the transaction time and the availability time are now-relative, is needed. In the literature, there are some proposals that support the now-relative transaction time, such as the Bitemporal R tree and the Double R (2R) tree [11]. Although the Bitemporal R tree is efficient for transaction time timeslice queries, because of its structure, it is not possible to add an additional now-relative time dimension, i.e. the availability time, to the Bitemporal R tree. As mentioned in [11], a second method that is based on the Double R tree method, the 2R tree, is a good alternative to the Bitemporal R tree and can be extended to support the 4 time dimensions. Figure 10 represents the structure of the proposed TAR* tree.

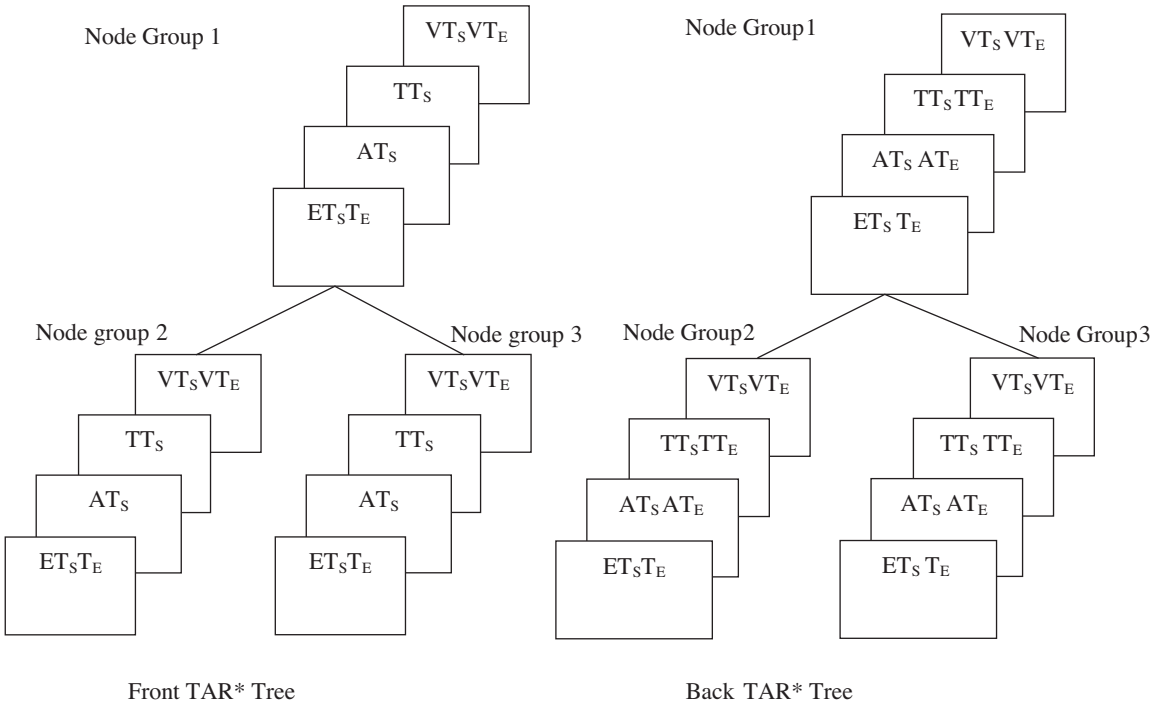

Figure 10. Structure of the TAR* tree.

As in the case of the 2R tree, there are 2 R trees in the TAR* tree structure. The front R tree holds the live objects and the back R tree stores the logically deleted objects. Both the front and back R trees support PD queries by using node groups. Data and their enclosing MBR (MBR will simply be used instead of the 4-D hyperrectangle throughout the paper) are represented by [VT_S, VT_E, ET_S, ET_E, TT_S, TT_E, AT_S, AT_E]. When a data item is to be inserted into the database (since it is live and the transaction and availability time end points are UC), it is inserted into the front R tree. If a data record is to be deleted (i.e. logically deleted) from the database, then the record is deleted from the front R tree and is inserted into the back R tree while setting its transaction end point to the deletion time and its availability time to a specific time that is provided by the user.

In the back R tree, data are indexed by 4 pairs of their time dimensions. However, in the front R tree, the transaction and availability time end points are UC and the front R tree indexes data objects according to their transaction and availability start points and valid and event time intervals. A data item and its enclosing

MBR are represented by [VT_S, VT_E, ET_S, ET_E, TT_S, AT_S] in the front R tree of the TAR* tree. The original 2R tree uses only the valid and transaction times, whereas a data item is indexed by its transaction start time point in the front R tree. In the TAR* tree, the valid time and transaction time dimensions are extended with the event and availability time dimensions.

When an object is inserted into the database, it is inserted into the front R tree. As explained for the AR* tree [12], the insertion algorithm is a naive extension of the original R* tree. The insertion algorithm takes the data object and a pointer to its data page and returns the MBR id of the data object in the front R tree. An important point in the TAR* tree structure is that the front R tree indexes the data objects using only the start points of their availability and transaction time dimensions. In order to delete a data object, i.e. logical deletion, 2 algorithms are invoked. The first step is to find the data record that will be deleted from the database in the front R tree. Since only logical deletion is supported in temporal databases, only live records can be deleted and all of the live records are stored in the front R tree. The search algorithm is invoked to find the data object in the R tree. Next, the data object is deleted from the front R tree and is inserted into the back R tree. The end point of the transaction time of the data object is set to the deletion time and the end point of the availability time is set to a specific time, which is obtained from the user. If the user does not supply the availability time, then the availability time is also set to the deletion time. The algorithm returns the MBR id of the data object in the back tree. As a result, the deletion operation is equivalent to deletion from the front R tree and insertion into the back R tree. The deletion algorithm that is used in the front R tree is also a naive extension of the original R* tree, as explained for the AR* tree [12]. The combination of 4 different time dimensions produces 2^4-1 query types. The node group structure of the AR* tree provides efficient processing for each query type. When a query that does not have any constraints in the transaction time and the end point of the availability time is asked, it is executed on the live objects in the front R tree (since the query execution time is *now*). All of the live objects are stored in the front R tree so that only the front R tree is used in a search with a query that does not have any constraint in the transaction time dimension. The search algorithm of the TAR* tree is similar to that of the AR* tree. However, a query that has a time constraint on the end point of the availability time or transaction time dimension is about logically deleted objects. Consequently, the query is executed on the back R tree, which stores only the logically deleted records. All of the remaining query types are executed in the same way and have to be processed in both the front and back R trees. The execution of retrieval queries in the front and back R trees is somewhat different. In the back R tree, queries are executed in the usual way, whereas in the front R tree, the transaction time and availability time constraints have to be transformed before evaluating. For example, if a query asks for data objects that are alive at t_i and valid at v_j (t is in the x-axis and v is in the y-axis), both the front and back R trees are searched. The back R tree is searched for all of the rectangles that contain the point (t_i, v_j). The front R tree is searched for all of the vertical intervals that intersect a horizontal interval H. Interval H starts from the beginning of the transaction time and extends until point t_i at height v_j. The advantage of the search algorithm in both R trees is that it does not access the irrelevant dimensions while searching, as is the case with the AR* tree. The search algorithm is the same as that for the AR* tree. The Transform Query algorithm is used for the front R tree. Only the start point of the transaction and availability time dimensions are indexed. Thus, when a query asks for a time point t_i in the transaction/availability time, the search has to start from the beginning of the transaction time and extend until point t_i. The Transform Query algorithm assigns the beginning of transaction time to TT_s and t_i to TT_E. The General Search algorithm for the TAR* tree is the same as for the AR* tree search algorithm.

Search Algorithm:
Input: rectangle: query range
node-group: initial node-group of the query
Output: result: all of the tuples in rectangle
Begin
If (initial node group contains TT_E and/or AT_E)
    generalSearch (rectangle, node-group, back R tree);
Else If (initial node group does not contain TT or AT)
    generalSearch (rectangle, node-group, front R tree);
Else If (initial node group contains TT_S and/or AT_S)
    generalSearch (rectangle, node-group, back R tree);
    rectangle ← –TransformQuery(TT_S and/or AT_S)
    generalSearch (rectangle, node-group, front R tree);
End
TransformQuery Algorithm:
Input: time point (t_i) for transaction time and/or time point (t_j) for availability time
Output: start and end point of transaction time and/or start and end point of availability time
Begin
If (TT_S)
    $TT_E \leftarrow -t_i$
    $TT_S \leftarrow$ – beginning of transaction time in the tree
If (AT_S)
    $AT_E \leftarrow -t_i$
    $AT_S \leftarrow$ – beginning of transaction time in the tree
End

The TAR* tree is designed for efficiently querying now-relative PD temporal range queries in temporal XML documents. We use the TAR* tree for 2 different reasons in our system. First, we use the TAR* tree to index the root-to-leaf paths. The number of TAR* trees needed is equal to the number of the root-to-leaf paths. Second, we use a single TAR* tree to index all of the nonleaf nodes. We have stated that the leaf nodes of the TAR* tree are composed of a pointer to the data and its enclosing MBR. When a new element is inserted into one of the TAR* trees, then its enclosing MBR is also inserted into the path index tables and the join index table. The MBR relation table is updated with the new MBR.

MBR relation table

The TAR* tree is used for indexing the temporal dimensions of the paths in a temporal XML document. However, the TAR* tree only indexes the temporal dimensions. If a query with a constraint on any of the key dimensions is asked (e.g., return all D nodes where D.Aid = 5 and D.Timestamp.ValidTime = 2), and if we use TAR* trees to execute the query, then unnecessary nodes would be accessed. As a solution, we can add the node identifiers as a new dimension to the TAR* tree. However, a path may contain multiple intermediate nodes, and since we do not know which node type's constraint will be given in a query, we have to add all of the intermediate node types as a key dimension to the tree. If we assume that the root-to-leaf path length is

n, this will lead us to add n different key dimensions to the TAR* tree. In the first case, if 2 nodes have the same timestamp values, they will be indexed in the same MBR. However, in the second case, the 2 nodes will be indexed in different boxes because of their key dimensions, and this will lead to an inefficiency in the TAR* tree searches.

A possible solution for this type of question is to not use the TAR* tree directly. Instead of searching the TAR* tree using only the time dimensions, we can search the path index and join index tables and apply the time constraints in those tables. The MBR fields in the path index tables and join index tables give the information of the temporal dimensions of the nodes. However, this does not prevent us from comparing the list of MBRs, because the records in the index tables store only the minimum bounding boxes of themselves or their descendants. There is no information on the relationships among the minimum bounding boxes. A good solution is to precompute the overlapping MBRs when a new node is inserted into the TAR* trees. This will prevent us from comparing the time dimensions during query processing.

We propose an MBR relation table to store the overlapping MBRs from all of the TAR* trees. The Table shows the structure of the MBR relation table. The TAR* Tree Path Type shows the root-to-leaf path type of the TAR* tree to which the MBR belongs. MBRId is the unique identifier of the MBR. "Overlapped MBRs" is the MBR ids in different TAR* trees that overlap with the given MBR id.

Table. MBR relation table.

TAR* Tree Path Type	MBRId	Overlapped MBRs
Root/.../Leaf1	MBR1	{Root/.../leaf2⇒MBR2,MBR4}
Root/.../Leaf2	MBR2	{Root/.../Leaf1⇒MBR1}, {All TAR*⇒MBR5}
All TAR* Tree	MBR6	{Root/.../Leaf2⇒MBR3}

An important property of temporal XML documents is consistency. As we stated earlier, the node time interval is the union of its children's time intervals. Thus, when a leaf node is inserted into its corresponding TAR* tree, it is obvious that its MBR will be overlapped with the MBR of its ancestors in the TAR* tree of the nonleaf nodes. Although the MBR relation table has the advantage of storing precomputed overlapping intervals and consequently increasing the search performance, it has the disadvantage of incurring extra update costs.

4. Performance study

We implemented and tested the proposed system and compared it with 3 other systems for the retrieval time of various queries. The proposed system contains path indices and a temporal multidimensional index that is used for temporal queries. We compared the proposed system with 2 different multidimensional indexing techniques. Moreover, we compared the proposed system with a temporal XML index, TempIndex [20], that was proposed in the literature.

The first structure that we use in our comparison is the Double R tree, which is designed for now-relative bitemporal data. The Double R tree is designed for fully dimensional queries. However, the system that we propose supports both fully and partially dimensional range queries. Thus, our proposed system clearly has a better performance for PD range queries than the Double R tree.

The second structure that we use in our comparison is the AR* tree, which is designed for PD range queries. While implementing the AR* tree, we use a maximum timestamp approach for now-relative timestamps. Our proposed system retrieves the results in a shorter time than the AR* tree does.

Our proposed system can be used for evaluating different kinds of query types, such as Selection, Join, Temporal Projection, Temporal Slicing, Temporal Join, and Temporal Partially Dimensional Range queries. In

the experiments, we focused on Temporal Partially Dimensional Range queries for the live and logically deleted data to test the performance of the TAR* tree.

We compared the TAR* tree with the AR* tree and the Double R tree. We tested the execution time of the Temporal Slicing and Temporal Partially Dimensional Range query types in the experiments. In order to measure the advantage of the proposed structure over the Double R tree methodology, we use partially dimensional range queries with 1, 2, 3, and 4 dimensions.

We have conducted 2 different sets of experiments. We used anaesthesia documents of 40 KB and 250 KB in size, respectively, in the first and second sets of experiments. The data structures that are proposed and compared in this paper were implemented in Java. The first set of experiments (for the 40-KB anaesthesia document) was tested on an Intel(R) Pentium(R) M processor 1.80 GHz PC, with 4 GB RAM, running under Windows XP. The second set of experiments (for the 250-KB anaesthesia document) was tested on an Intel Core i5 processor 2.30 Ghz PC, with 4 GB RAM, running under Windows 7. The anaesthesia datasets are generated from a sample anaesthesia record. The anaesthesia records that are used in anaesthesia information systems in healthcare institutions do not have a standard format but have a similar structure. We used 3 different anaesthesia record formats based on the Central Vermont Medical Center anaesthesia records [31]. We generated 9000 anaesthesia documents for the first set of experiments and 3000 anaesthesia documents for the second set. All of the documents are based on the 3 different anaesthesia formats. All of the time and other element values are randomly generated. Each element has 4 types of time values that are also randomly generated. The dataset contains live and deleted data. The exact amount of deleted data depends on the path, but it is about 6% of the live data for the AgentTypeAmount element. The anaesthesia documents are composed of 3 periods: preoperative, intraoperative, and postoperative. In the first set of experiments, there are 76 different root-to-leaf paths in each document. One of the root-to-leaf paths is an AgentType element, which is the descendant of the case data in the intraoperative period (patient/surgery/intraoperativePeriod/CaseData/GasesAndAgents/Agent/AgentTypeAmount).

We filter the amount of AgentType elements in the queries. In each of the anaesthesia documents, there are 20 and 148 AgentTypeAmount elements, respectively, in the first and second sets of experiments.

In the intraoperative period, the anaesthesia data are recorded every 2 to 5 min. Thus, we use the anaesthesia datasets in which the time granularity of the anaesthesia data is in "minutes". The measurements are done in terms of the query processing time. In each experiment, we calculated the percentage of the proposed system's performance gain over the compared system. In each experiment, each query is run 10 times with differing values, and the mean is used in Figures 11–19. The queries used in all of the experiments are the same and are given in the following. Each query is run for live, dead, or both live and dead data.

1 dim VT:
List the drug amounts that are valid during 10.12.2006 18.35 - 10.12.2006 22.00.
patient/surgery/intraoperativePeriod/CaseData/Drugs/Drug/DrugAmount/TimeElement/VT [Low $\Rightarrow$ 10.12.2006 18.35 and High $\Leftarrow$ 10.12.2006 22.00]
1 dim AT:

List all amounts of live agent types (from gases and agents) from intraoperative period whose availability time starts between 12.10.2006 20.20 and 12.10.2006 21.20?
/patient/surgery/intraoperativePeriod/CaseData/GasesAndAgents/Agent/AgentTypeAmount/TimeElement/AT[low $\Rightarrow$ 12.10.2006 20.20 and low $\Leftarrow$ 12.10.2006 21.20 and high = UC]

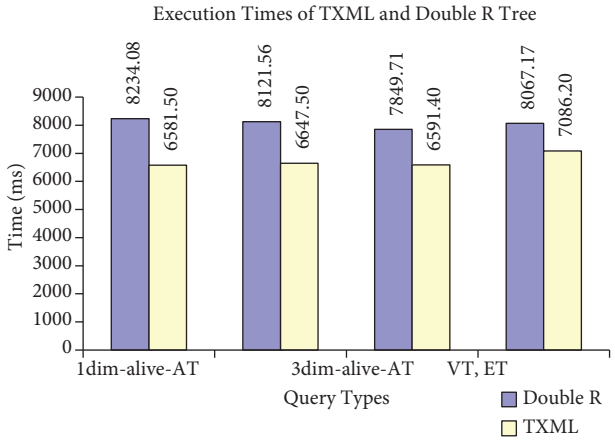

Figure 11. Query execution times of TXML and Double R tree for the 40-KB anaesthesia documents.

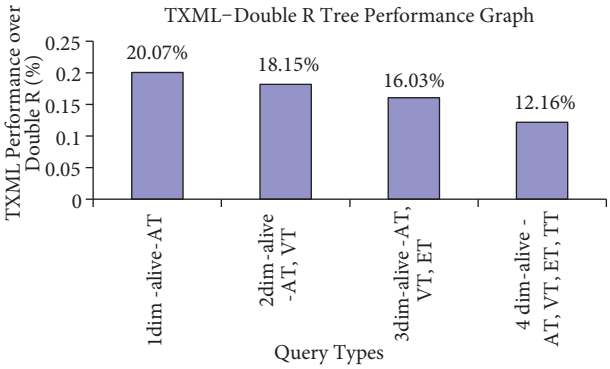

Figure 12. TXML–Double R tree performance graph for the 40-KB anaesthesia documents.

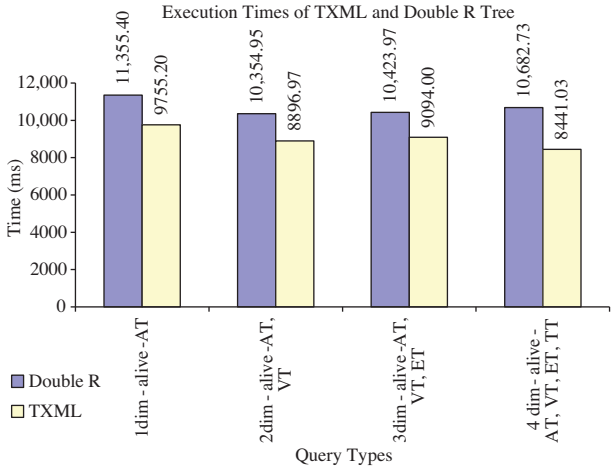

Figure 13. Query execution times of the TXML and Double R tree for the 250-KB anaesthesia documents.

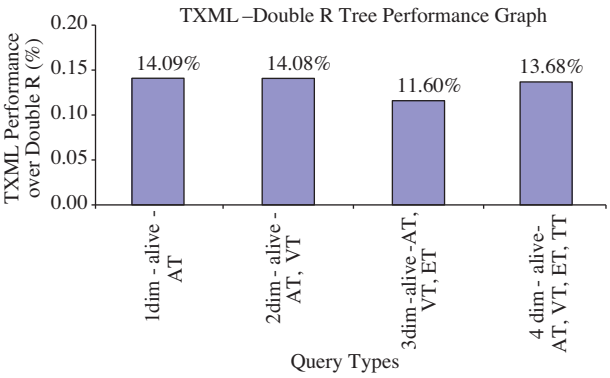

Figure 14. TXML–Double R tree performance graph for the 250-KB anaesthesia documents.

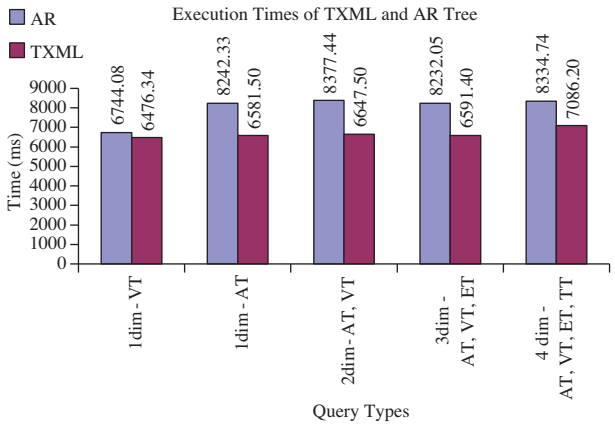

Figure 15. Query execution times of the TXML and AR tree for the 40-KB anaesthesia documents.

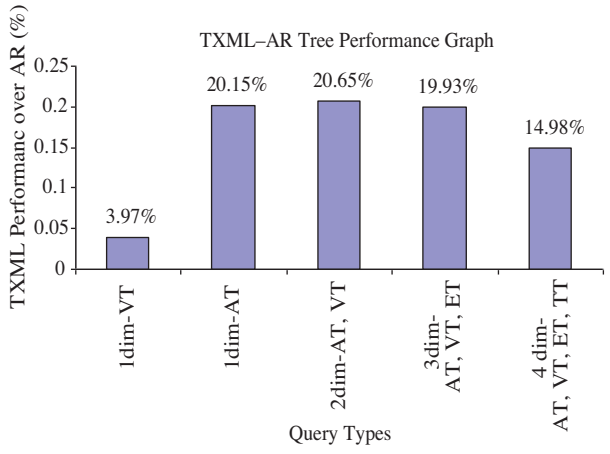

Figure 16. TXML–AR* tree performance graph for the 40-KB anaesthesia documents.

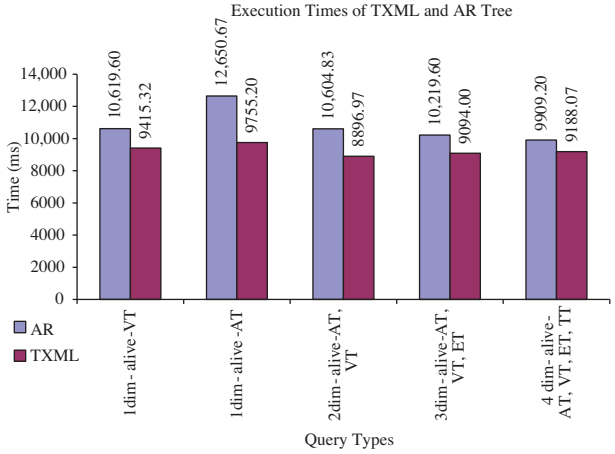

Figure 17. Query execution times of the TXML and AR tree for the 250-KB anaesthesia documents.

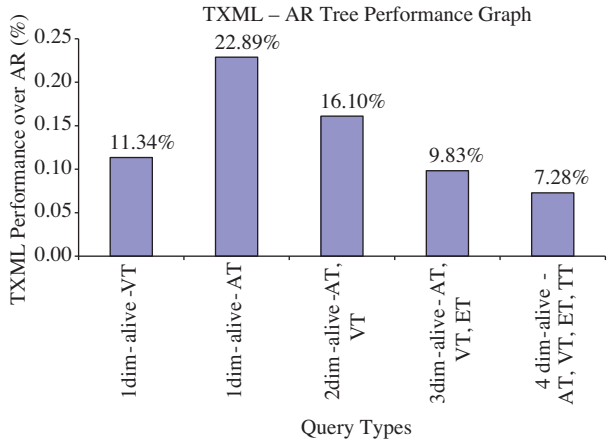

Figure 18. TXML–AR* tree performance graph for the 250-KB anaesthesia documents.

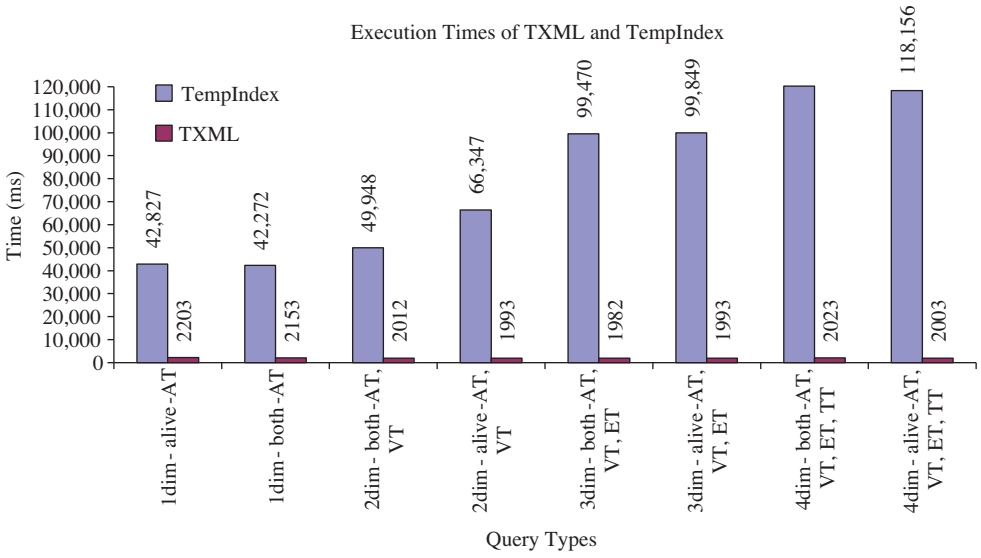

Figure 19. Query execution times of TXML and TempIndex for the 40-KB anaesthesia documents.

2 dim AT, VT:

List all amounts of live agent types (from gases and agents) from intraoperative period whose availability time starts between 12.10.2006 20.20 and 12.10.2006 21.20 and validity time starts after 12.10.2006 20.20 and ends before 12.10.2006 21.20?
/patient/surgery/intraoperativePeriod/CaseData/GasesAndAgents/Agent/AgentTypeAmount/TimeElement
[AT/low $\Rightarrow$ 12.10.2006 20.20 and AT/low $\Leftarrow$ 12.10.2006 21.20 and AT/high = UC and VT/low $\Rightarrow$ 12.10.2006 20.20 and VT/high $\Leftarrow$ 12.10.2006 21.20]

3 dim AT, VT, ET:

List all amounts of live agent types (from gases and agents) from intraoperative period whose availability time starts between 12.10.2006 20.20 and 12.10.2006 21.20 and validity time starts after 12.10.2006 20.20 and ends before 12.10.2006 21.20 and event time starts after 12.10.2006 20.20 and ends before 12.10.2006 21.20?

/patient/surgery/intraoperativePeriod/CaseData/GasesAndAgents/Agent/AgentTypeAmount/TimeElement [AT/low ⇒ 12.10.2006 20.20 and AT/low ⇐ 12.10.2006 21.20 and AT/high = UC and VT/low ⇒ 12.10.2006 20.20 and VT/high ⇐ 12.10.2006 21.20 and ET/low ⇒ 12.10.2006 20.20 and ET/high ⇐ 12.10.2006 21.20]

4 dim AT, VT, ET, TT:

List all amounts of live agent types (from gases and agents) from intraoperative period whose availability time starts between 12.10.2006 20.20 and 12.10.2006 21.20 and validity time starts after 12.10.2006 20.20 and ends before 12.10.2006 21.20 and event time starts after 12.10.2006 20.20 and ends before 12.10.2006 21.20 and which is recorded (transaction time) between 12.10.2006 20.20 and 12.10.2006 21.20?

/patient/surgery/intraoperativePeriod/CaseData/GasesAndAgents/Agent/AgentTypeAmount/TimeElement [AT/low ⇒ 12.10.2006 20.20 and AT/low ⇐ 12.10.2006 21.20 and AT/high = UC and VT/low ⇒ 12.10.2006 20.20 and VT/high ⇐ 12.10.2006 21.20 and ET/low ⇒ 12.10.2006 20.20 and ET/high ⇐ 12.10.2006 21.20 and TT/low ⇒ 12.10.2006 20.20 and TT/low ⇐ 12.10.2006 21.20 and TT/high = UC]

Figures 11 and 13 show the execution times of the queries using the TAR* tree and the Double R tree. We also present the performance comparison of the proposed TAR* tree over the Double R tree for the fully and partially dimensional queries on the live data in Figures 12 and 14. The only difference in the proposed system and the compared system is the time index structure. We focused on the queries that filter live data because both the TAR* tree and the Double R tree have special structures for live and deleted data. As the number of query dimensions gets closer to the number of data dimensions, we expect that the performance of the proposed model will get closer to the Double R, since Double R structure is designed for fully dimensional queries. The experiment shows that the proposed tree (TAR* tree) has better performance in all of the PD temporal queries over the Double R tree. However, as the query dimension increases, the performance of the TAR* tree gets closer to the Double R tree. This is because the search region can be limited in the case of large number of dimensions.

To compare the performance of the TAR* tree over the AR* tree, we execute partially and fully dimensional temporal queries. In our proposed system, the valid time and event time dimensions are bounded, i.e. they do not contain now-relative timestamps. In the AR* tree, the now-relative timestamps are inserted using the maximum timestamp approach and the bounded timestamps are inserted according to their values. On the other hand, in the TAR* tree, the now-relative timestamps are inserted as points and the bounded timestamps are inserted according to their values.

In the AR* tree, logically deleted and live data are stored in the same structure. However, in the TAR* tree, live data and deleted data are stored in different trees to decrease the search space.

Figures 15 and 17 show the execution times of the TAR* tree over the AR* tree and Figures 16 and 18 show the performance gain of the TAR* tree over the AR* tree. Because of the maximum timestamp approach, we expect the performance of the TAR* tree to be better than that of the AR* tree for data. The experiments show that the TAR* tree has better performance than the AR* tree for every temporal query. When we compare the performance of the proposed tree with the AR* tree for now-relative timestamps (1 dim-AT) and bounded timestamps (1 dim-VT), the experiments show that for the now-relative timestamps, the proposed tree has better performance than that for the bounded timestamps. In our datasets, the percentage of the deleted elements is approximately 6% of the AgentTypeAmount elements. We claim that as the percentage of the deleted elements increases, the performance of the TAR* tree over the AR* tree also increases.

The last important system that we compare our system with is the TempIndex. The TempIndex is for

tracking historical information in an XML document and for recovering the state of the document for any given time [20]. It uses continuous paths, which are valid continuously during a certain interval in a temporal XML graph (used for summarising and indexing temporal XML documents). For indexing continuous paths, different data structures have been proposed in the TempIndex (i.e. a new class of summaries called TSummary that adds the time dimension to the path summarisation schemes).

We compared our proposed system and the TempIndex in terms of temporal retrieval queries. The TempIndex is proposed for 1 temporal dimension, whereas our proposed system supports 4 different temporal dimensions. To support 4 temporal dimensions in TempIndex, we implemented temporal depth tables for each temporal dimension.

In TempIndex, the temporal filtering function uses temporal depth tables. It scans the appropriate temporal depth table sequentially. If the query has multiple criteria on different temporal dimensions, then multiple temporal depth tables are scanned. Each temporal depth table contains all of the nodes in the dataset with the same temporal depth. On the other hand, our proposed system uses specialised temporal data structures (TAR* tree) for filtering the temporal data, which have better performance over the sequential search.

We tested and compared the 2 systems with 500 anaesthesia records of 40 KB in size. Figure 19 shows the execution times of queries using TXML and TempIndex. TXML executes the specified queries considerably faster than TempIndex. For 9000 documents, the TempIndex implementation took too long to respond. The main reason for the short execution time of our proposed method is the performance of the TemporalAR* tree over the sequential search in the temporal depth tables. The experiment shows that proposed system has an increasing performance on multiple time dimensions. As the number of dimensions increases in the query, the number of temporal depth tables also increases, since we store each temporal dimension in a different temporal depth table. As a result, more tables are sequentially searched, which increases the execution time for TempIndex.

TXML has indices for processing nontemporal data, as well. In Figure 20, we show the execution time of the nontemporal selection Query 7: List the patients of surgeon "J. Smith". //patient/ surgery/intraoperative/surgeons [primary="J.Smith"]. Question 7 is applied to the 40-KB anaesthesia documents.

In Figure 21, we show the execution time of another nontemporal query, Query 8, which combines selection and join operations. Query 8 is applied to the 250-KB anaesthesia documents. Query 8 lists the

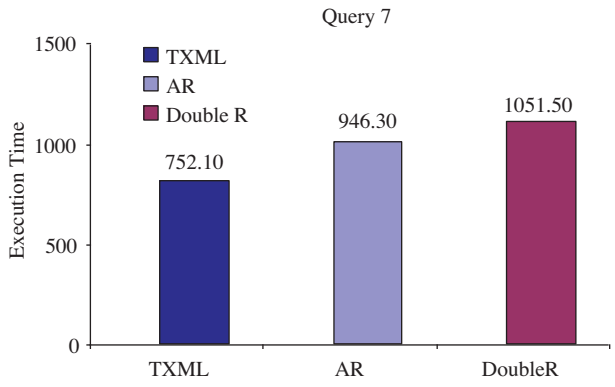

Figure 20. Execution time (in ms) of Query 7, a nontemporal selection query, for the 40-KB anaesthesia documents.

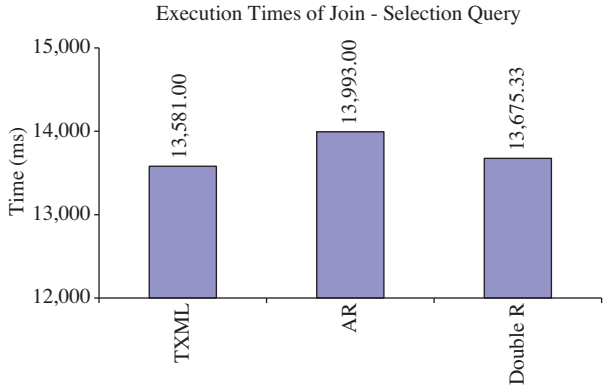

Figure 21. Execution time of Query 8, a nontemporal join query, for the 250-KB anaesthesia documents.

LactatedRingsType Amounts of patients to whom EPHEDRINE was applied. For this query, the following 2 paths are joined according to the Case Data element.

"patient/surgery/intraoperativePeriod/CaseData/Drugs/Drug/DrugName['EPHEDRINE']";

"patient/surgery/intraoperativePeriod/CaseData/LactatedRings/LactatedRingsTypeAmount".

The AR* tree and the Double R tree had to be augmented with path index and join index to be able to process the nontemporal and some temporal queries. Although all 3 models use similar structures to answer Query 7, the structures contain time data (i.e. other storage structures are accessed) and this affects the performance of the nontemporal queries. TXML performs 20.52% better than the AR* tree and 28.47% better than the Double R tree. In this experiment, we have 2100 XML anaesthesia documents.

The TXML storage structures are designed not only for expediting the processing of various types of temporal queries (on VT, AT, ET, TT), but at the same time that of nontemporal and mixed temporal and nontemporal queries. In our design, we consider the fact that secondary storage space is rather inexpensive and the bottleneck in the performance is the time spent in accessing data [11]. Thus, our design is geared towards expediting retrieval. Due to the presence of various index structures in the proposed method, a considerable amount of secondary storage space is used. None of the index structures compared in our experiments have the capabilities that TXML possesses. To perform our experiments and realise the given comparisons, we had to augment the mentioned index structures with extra storage structures. After this augmentation in our implementation for the experiments, the secondary storage space used by TXML and the Double R tree is almost the same. The storage space used by the AR* tree is nearly 10% more than that used by TXML, whereas the space used by TempIndex is nearly 2.5 times more than that used by TXML. The most space consumption (around 5 GB in the first set of experiments) is due to the MBR index, which exists in the Double R and AR* trees. In TempIndex, there is no MBR, but the Temporal Depth structure that is used instead takes up huge amounts of space (several times that of the MBR index). Storage space requirements of just the indices primarily used for time, i.e. the TAR index, AR index, Double R index, and tMapIndex in TXML, the AR* tree, the Double R tree, and TempIndex, respectively, are close (around 450 MB in the first set of experiments) in our implementation for the experiments.

5. Conclusion

The time information in anaesthesia data is important because of research, audit, and medicolegal issues. Most of the current temporal XML models only record the valid time and transaction time. However, event and availability times are also needed in order to represent time information accurately and to be able to make some critical decisions. In this paper, we proposed a temporal XML data model, TXML, for anaesthesia data, which records the valid time, transaction time, event time, and availability time, in order to provide the temporal information of the events associated with anaesthesia. TXML can be used for any health-related or otherwise temporal XML data that contain the transaction time, bounded valid time, event time, and availability time information.

The main contributions of the paper can be summarised as follows: attaching time information to XML-based anaesthesia records, recording 4 different time dimensions of anaesthesia data, and efficiently processing partially and fully dimensional temporal queries.

TXML consists of different index structures to improve the performance of the system by decreasing the number of page accesses. Two different index types, PathIndex and JoinIndex, are proposed in the system in order to support path indexing. PathIndex is used for indexing root-to-leaf paths in the system and JoinIndex

is used for nonleaf paths. Nontemporal queries can be efficiently processed by the path index and join index structures without having to traverse the anaesthesia document tree.

In order to process temporal queries, a new temporal index structure, the TAR* Index, is proposed. The TAR* Index makes use of the advantages of the AR* tree for partially dimensional range queries. It also makes use of the advantages of the Double R tree and stores live and logically deleted records in 2 different trees to avoid large rectangle formation, consequently avoiding excessive dead spaces and overlaps for now-relative data. Combining the advantages of the AR* tree and the Double R tree, the TAR* tree helps to efficiently process partial and full dimensional temporal queries.

Supporting 4 different time dimensions allows inquiring on different types of temporal and nontemporal queries. The model's implementation is not only efficient in processing temporal anaesthesia queries (such as temporal projection, temporal selection, temporal slicing, temporal join, time period containment, and temporal comparison) but is also efficient for nontemporal anaesthesia queries that are used in anaesthesia audits or in scientific research.

References

[1] V. Parmar, "Role of information management system and automated record in anaesthesia", Indian Journal of Anaesthesia, Vol. 50, pp. 99–102, 2006.

[2] M. Gardner, T. Peachey, "A standard XML schema for computerized anaesthetic records", Anaesthesia, Vol. 57, pp. 1174–1182, 2002.

[3] HL7 SIGGAS. Generation of Anaesthesia Standards, available at http://www.hl7.org, 2005.

[4] Y. Shahar, "Timing is everything: temporal reasoning and temporal data maintenance in medicine", Proceedings of the Conference on Artificial Intelligence in Medicine and Medical Decision Making, pp. 30–46, 1999.

[5] J.M. Feldman, "Medicolegal aspects of anaesthesia information management systems", Seminars in Anaesthesia, Perioperative Medicine and Pain, Vol. 23, pp. 92–86, 2004.

[6] M. Böhlen, J. Clifford, R. Elmasri, S.K. Gadia, F. Grandi, P. Hayes, "The consensus glossary of temporal database concepts - February 1998 version", in Temporal Databases: Research and Practice, Springer, Berlin, pp. 376–405, 1997.

[7] C.S. Jensen, "Introduction to temporal database research", in Temporal Database Management, thesis, available at http://www.cs.aau.dk/~csj/Thesis/pdf/chapter1.pdf, 2000.

[8] B. Stantic, G. Governatori, A. Sattar, "Handling of current time in native XML databases", Proceedings of the 17th Australasian Database Conference, Vol. 49, pp. 175–182, 2006.

[9] S.K. Kim, S. Chakravarthy, "Modelling time: adequacy of three distinct time concepts for temporal databases", Proceedings of the 12th International Conference on the Entity-Relationship Approach, pp. 475–491, 1993.

[10] C. Combi, A. Montanari, "Data models with multiple temporal dimensions: completing the picture", Proceedings of the 13th Conference on Advanced Information Systems Engineering, pp. 187–202, 2001.

[11] A. Kumar, V.J. Tsotras, C. Faloutsos, "Designing access methods for bitemporal databases", IEEE Transactions on Knowledge and Data Engineering, Vol. 10, pp. 1–20, 1998.

[12] Y. Feng, A. Makinouchi, "Efficient evaluation of partially-dimensional range queries using adaptive R*-tree", Proceedings of the 17th International Conference on Database and Expert Systems Applications, Vol. 4080, pp. 687–696, 2006.

[13] N. Beckmann, H.P. Kriegel, R. Schneider, B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles", Proceedings of the ACM SIGMOD International Conference on Management of Data, Vol. 19, pp. 322–331, 1990.

[14] Y. Cui, "High dimensional indexing", Lecture Notes in Computer Science, Vol. 2341, 2003 (monograph).

[15] J. Balust, A. Macario, "Can anesthesia information management systems improve quality in the surgical suite?", Current Opinion in Anesthesiology, Vol. 22, pp. 215–222, 2009.

[16] A. Meyer-Bender, R. Spitz, B. Pollwein, "The anaesthetic report: custom-made printouts from anaesthesia-information-management-systems using extensible stylesheet language transformation", Journal of Clinical Monitoring and Computing, Vol. 24, pp. 51–60, 2010.

[17] F. Wang, F.C. Zaniolo, "XBIT: an XML-based bitemporal data model", Lecture Notes in Computer Science, Vol. 3288, pp. 824–810, 2004.

[18] F. Grandi, F. Mandreoli, P. Tiberio, "Temporal modeling and management of normative documents in XML format", Data and Knowledge Engineering - Special Issue: WIDM, Vol. 54, pp. 354–327, 2003.

[19] A. Vaisman, A.O. Mendelzon, E. Molinari, P. Tome, "Temporal XML: data model, query language and implementation", University of Toronto, available at http://www.cs.toronto.edu/~avaisman/papers.html, 2004.

[20] F. Rizzolo, A.A. Vaisman, "Temporal XML: modeling, indexing and query processing", The International Journal on Very Large Data Bases, Vol. 17, pp. 1179–1212, 2008.

[21] M. Duong, Y. Zhang, "LSDX: a new labeling scheme for dynamically updating XML data", Proceedings of 16th Australasian Database Conference, pp. 185–193, 2005.

[22] Q. Li, B. Moon, "Indexing and querying XML data for regular path expressions", Proceedings of the 27th International Conference on Very Large Data Bases, pp. 361–370, 2001.

23 E. Cohen, H. Kaplan, T. Milo, "Labeling dynamic XML trees", Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 271–281, 2002.

[23] The Apache XML Project, About Apache Xindice, available at http://xml.apache.org/xindice/, 2006.

[24] Sedna, About Sedna, Native XML Database System, available at http://modis.ispras.ru/sedna/index.htm, 2006.

[25] Data ex Machina, The Natix XML Repository History, available at http://www.dataexmachina.de/natix.html, 2006.

[26] Timber, Introduction, available at http://www.eecs.umich.edu/db/timber, 2006.

[27] K.M. Win, W.K. Ng, E.P. Lim, "ENAXS: efficient native XML storage system", Proceedings of the 5th Asia-Pacific Web Conference on Web Technologies and Applications, Vol. 2642, pp. 59–70, 2003.

[28] A. Guttman, "R-Trees - a dynamic index structure for spatial searching", Proceedings of the ACM SIGMOD International Conference on Management of Data, Vol. 14, pp. 47–57, 1984.

[29] R. Bliujute, C.S. Jensen, S. Saltenis, G. Slivinskas, "R-Tree based indexing of now-relative bitemporal data", Proceedings of the 24th International Conference on Very Large Data Bases, pp. 345–356, 1998.

[30] Rapid Record, A Computer Assisted Anaesthesia Record Keeper, available at http://www.rapid-record.com/index.htm.