A TEMPORAL XML DATA MODEL AND ITS IMPLEMENTATION IN ANAESTHESIA INFORMATION SYSTEMS

by
Oya Ünlü
B.S., Computer Engineering, Boğaziçi University, 2004

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2007

ACKNOWLEDGMENTS

I would like to thank many people who have helped and given support during my thesis research.

First of all, I would like to thank my supervisor Prof. Taflan Gündem for his invaluable guidance and advises during both this research and my academic life. His valuable comments and ideas helped me to improve this thesis.

I would also like to thank Assist. Prof. Ayşe Başar Bener and Assist. Prof. Ata Akın, for their participations in my thesis committee and their valuable comments. I would like to mention their help to find sample anaesthesia data for my research.

Special thanks to Burak Duygulu, for his valuable comments during the implementation phase. I would also thank him for his support and understanding throughout my study. His encouragement and patience helped me to complete this research.

And finally, I would like express my gratitude to my parents and grandmother for their both moral and material support and patience during my academic life and this research. They were my first teachers and helped me to shape my life with their invaluable ideas. I owe them a debt of thanks for everything in my life.

This thesis is dedicated to my parents for their endless love and support throughout my life.

ABSTRACT

A TEMPORAL XML DATA MODEL AND ITS IMPLEMENTATION IN ANAESTHESIA INFORMATION SYSTEMS

Information systems are widely used in health domain. In today's world, medical records, hospital information systems, anesthesia information systems are some of the crucial parts of the health systems.

Time is very important in health information systems. From the point of medical research, auditing and medico legal aspects, without the time in health information systems such as electronic medical records, decision support systems and anesthesia records, information can not be represented and queried correctly. In the literature, valid and transaction times are the most frequently used temporal dimensions. However decision and availability times are also needed in order to store and represent information accurately.

In today's healthcare institutions data are stored in variety of systems. As an emerging technology, XML solves most of the problems such as data exchange and different structures in health information systems.

The main contributions of this thesis are summarized as follows. Combining the requirements of XML and time, we propose a temporal XML data model for anaesthesia information systems with four different temporal dimensions: valid time, transaction time, event time and availability time. A new storage structure and a temporal index structure are proposed in order to store temporal XML data and process queries efficiently.

A variety of queries are tested on the proposed model and two different alternative systems which are designed for partially dimensional queries and now relative bitemporal data. The experiments show that the proposed model has better performance for temporal partially dimensional range queries and now relative data.

ÖZET

ANESTEZİ BİLGİ SİSTEMLERİNDE ZAMANSAL XML TABANLI VERİ MODELİ

Bilgi sistemleri sağlık alanında yaygın olarak kullanılmaktadır. Günümüzde elektronik hasta kayıtları, hastane bilgi sistemleri ve anestezi bilgi sistemleri sağlık sistemlerinin vazgeçilmez parçalarıdır.

Sağlık sistemlerinde saklanan veriler zamanla birlikte değişebilir. Bilimsel araştırmalar, denetimler ve tıbbi ve hukuki açıdan, elektronik hasta kayıtları, karar destek sistemleri ya da anestezi kayıtları gibi sağlık sistemlerinde, veriler zamana bağımlı olmalıdır. Literatürde tanımlanan zaman boyutlarından en çok kullanılanları, geçerlilik zamanı ve işlem zamanıdır. Bununla birlikte, verilerde oluş zamanı ve öğrenilme zamanı tutulmazsa, bilgi doğru olarak saklanamaz ve sorgulanamaz.

Bugünkü sağlık kuruluşları, veri transferindeki zorlukları ve yapı problemlerini çözümlediği için gelişen bir teknoloji olan XML tabanlı sistemleri kullanmaktadır.

Bu çalışmada XML'in ve zamanın özelliklerini dikkate alarak, anestezi bilgi sistemleri için zamana bağımlı ve dört farklı zaman boyutunu destekleyen XML tabanlı bir veri modeli önerilmiştir. İkinci olarak, önerilen zamana bağımlı XML tabanlı model için, zamana bağımlı veriyi saklayan ve indeksleyen veri yapıları geliştirilmiştir. Son olarak, veriler üzerinde zamanla ilgili sorulan sorguların teklif edilen yapılarda hangi işlemlerle cevaplanabileceğini sunulmuştur.

Önerilen model, literatürde yer alan çok boyutlu zaman indeksleri ve kısmi sorgulama yapan indekslerle karşılaştırılmıştır. Test sonuçları, önerilen modelin zamana dayalı kısmi sorgularda diğer modellerden daha iyi sonuçlar verdiği göstermiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AT	Availability Time
AAR	Automated Anaesthesia Record
AR*-tree	Adaptive R*-tree
AIMS	Anaesthesia Information Management Systems
BH-document	Bitemporal History Document
DTD	Document Type Definition
DOM	Document Object Model
ET	Event Time
HL7	Health Level Seven
LSDX	Labeling scheme for dynamic XML
MBR	Minimum Bounding Region
SAX	Simple API for XML
TAR*-tree	Temporal Adaptive R*-tree
TS	Timestamps
TT	Transaction Time
TXML	The proposed model
UC	Until changed
VT	Valid Time
XBIT	An XML Based Bitemporal Data Model
XML	Extensible Markup Language
XPath	XML Path Language
XQuery	XML Query Language
W3C	World Wide Web consortium

1. INTRODUCTION

1.1. Motivation

Temporal databases are designed to record and query time-varying information efficiently. Most database applications are temporal in their nature, e.g.: financial applications, record-keeping applications, inventory management, scheduling applications and scientific applications.

Health databases contain a significant amount of temporal information. It is almost impossible to represent data and reason them without a temporal dimension in health domain, since information without temporal dimensions will be incomplete. A wide variety of applications need to deal with temporal aspects of health data. Such applications are time-oriented medical records of ambulatory or hospitalized patients, systems that predict future values of clinical data from past trends, temporal abstraction systems of clinical data, time-oriented knowledge-based decision-support systems such as systems supporting diagnosis, monitoring, therapy planning and anesthesia information systems. These time-oriented applications are used in most of the clinical areas such as cardiology, oncology, psychiatry, internal medicine, intensive care, cardiac surgery, orthopedics, urology, infectious diseases, pediatrics, endocrinology and anesthesiology.

There are multiple advantages of using temporal databases in health information systems, including the ability to answer both research and legal questions. Temporal data helps to clarify the answers of legal questions in some critical situations. For example when a doctor is accused of malpractice, a temporal database will accurately answer the question: “When the physician prescribed sulpha on October 14 2006, did she know at that time that the patient had an allergic reaction to sulpha on a previous date?” Moreover, illegal modifications on the records can be easily detected using temporal dimensions.

One of the important goals of the health information systems is to facilitate research on outcomes in the health domain. Temporal data models in electronic health systems

enable researchers to extract data values that follow certain simple temporal patterns or the relation between factors and the episodes.

Temporal data management is difficult in conventional, i.e. non-temporal, database systems and query languages. In order to support time related information in non-temporal databases, developers create application dependent solutions, which are usually ineffective and inefficient. These solutions must be reinvented each time when a new application is developed. The main difference between non-temporal and temporal databases is that, temporal databases manage time-oriented data in a systematic and efficient way.

In the literature, there have been considerable amount of research about temporal databases. In the early researches, relational data models were extended to support temporal dimensions. However the notion of time is difficult to represent in one single relational model. This limitation of relational databases had led researchers to use object-oriented databases. [3] [4] [13]

Temporal object-oriented data models can more accurately capture the semantics of complex temporal objects than relational data models and treat time as a fundamental component. Although object-oriented data models are more convenient, the emerging technology, XML, has been widely used in the researches for temporal data in the last years for two reasons: XML hierarchical structure can naturally represent the history of databases in a temporally grouped data model and powerful temporal queries can be expressed in XML based query languages without requiring any extension to current standards.

In today's large healthcare institutions, data is stored in different systems and in different models. Converting all the information systems into a single platform will be the best solution however it is impossible because of the need of huge capital investment. XML solves this problem because it eliminates the need of consolidating data into a single system by providing a standard means of communicating data between and among systems.

As mentioned in [6], Health Level Seven (HL7) is one of several American National Standards Institute accredited Standards Developing Organizations operating in the healthcare area. HL7 was founded to develop standards, which will be used by independent healthcare oriented computer systems. HL7 is working with XML technology and recommends on use of XML standards for all of HL7's platform and vendor-independent healthcare specifications. HL7 provides time data types for representing temporal dimensions. HL7 supports timestamps, time intervals and periodic time intervals as temporal dimensions.

Anesthesia Information Management Systems (AIMS) are one of the important parts of hospital information systems. It is a computer system that captures anesthesia-related information in a digital format and gathers information on every patient with whom a department of anesthesia interacts. The information includes data collected from pre-operative, intra-operative and post-operative phases. The anesthesia information is stored in a database where it can be arranged in a format for analysis. The AIMS database exchange data with other databases in the hospital or a larger health care delivery system. In the recent years, there is a research on the standards of XML Schema for computerized anesthesia records [7]. HL7 Anaesthesia Special Interest Group continues a research on the proposed Anaesthesia XML Schema, to make it compatible with HL7 standards.

Anesthesia information systems offer the ability to record data automatically during anesthesia care and to produce a complete, highly legible paper record for the patient's medical record. From the medico-legal aspects of the records, the electronic anesthesia record is the fundamental medico-legal document when allegations of negligent care are brought against the anesthesia care team. Temporal dimensions play an important role in the anesthesia information systems from the medico-legal point of view. Addition of temporal dimensions will help to expose any malpractice in anesthesia information systems.[8]

In this paper, we propose a temporal XML data model for anaesthesia information systems. Chapter 2 presents the background information about temporal concepts and their meaning in the health domain. The existing temporal XML data models and multidimensional index structures are investigated. In chapter 3, we present our temporal

XML data model for anaesthesia information systems. We propose physical storage structures and the index structures that are used for implementing temporal XML anaesthesia documents. In chapter 4, we present a performance evaluation of the proposed system. Finally, in chapter 5 we conclude the thesis with a summary of the main contributions.

2. BACKGROUND INFORMATION

In this chapter, we present the background information to understand the meaning of temporal concepts and their properties. We first give the definitions of the temporal concepts such as time instants, intervals and temporal elements. Secondly, we present the temporal dimensions that are defined in the literature and their meanings in the health domain. Thirdly, we classify the temporal query types and their usages. Fourthly, we present the temporal XML data models and the existing native XML databases and their support for the temporal dimensions. Finally, we investigate and classify the existing multidimensional index structures and present their advantages and disadvantages.

2.1. Introduction to Temporal Concepts

Information can change as the time progresses. Conventional databases are designed to capture the most recent data, i.e. current data. They cannot reflect the state of information at any given time because updates overwrite the previous state of data. So they capture a snapshot of reality. Although conventional databases serve some applications well, they are insufficient for the applications in which past or future data are also required. Storing only current data will result in a loss of information as times goes by. What is needed is a database that efficiently supports storing and querying of time varying-information. For this reason, it is essential to add temporal dimensions to the data model. However, in existing database technologies little support is given for managing temporal data. Time dimensions can be added to data models by defining user-defined times. However, usage of the user-defined temporal attributes have some disadvantages, such as users have to define all temporal attributes beforehand and they are responsible for maintaining the keys. Temporal databases support temporal dimensions using appropriate data structures and index structures for time-related data so that it can be efficiently and accurately stored and queried.

Several basic choices have to be made in order to model time-oriented data. Different time data types may be used for capturing the temporal aspects of data, including instants, time intervals, and temporal elements.

2.1.1. Time Instants and Intervals

An instant of time is defined as a point in the time axis in [9]. Time instants can be represented by natural numbers in discrete time models and by real numbers in continuous time models. For example an occurrence time of an event can be displayed by a time instant.

A time interval is a period between two time instants. It can be represented by a set of continues granules [9]. For example, validity of a fact can be defined by using a time interval.

Temporal granularity is described by a start point on the time axis plus the size of the each granule. Time granularities are used in specifying time intervals and instants. Different events require different levels of granularity. For example, the granularity of drug therapies can be given in “days” where time granularity of surgeries is in “minutes” [10].

Time instants and time intervals have been used in health domain to represent time-related data of instantaneous events such as myocardial infarction and events that are lasting for a period of time such as drug therapy. For the basic time entities in health domain, time instants are used. Time intervals of events are represented by their upper and lower temporal bounds, i.e. start and end time points.

In practice, health informatics application systems use both time point and time interval based approaches. In the health domain, HL7 provides different types of time data types, which use both time instants and time intervals [6]. Timestamps are represented by TS data type with varying precisions. TS data type is mostly used as calendar expressions.

For example, an anaesthesia drug dose was administered at 3pm on July 28, 2003 will be represented as:
<effectiveTime xsi:type=”TS” value=20030805”/>

HL7 IVL_TS data type is defined for representing time intervals. Indicating start and stop dates, the duration of an interval or the middle of the interval can be represented by IVL_TS data type using “high”, “low”, “highClosed”, “lowClosed”, “width”, “center” attributes.

A time interval starting from 1.12.2006 to 31.12.2006 can be represented as:

```
<effectiveTime xsi:type=”IVL_TS”>
    <low value=”20030707”/>
    <high value=”20030711”/>
</effectiveTime>
```

PIVL_TS is another data type to express periodic expressions. The data type supports periodic activities with a declared duration. PIVL_TS is a complex XML element that includes a period for representing the length of the time between individual acts, a phase for representing the length of an individual act and an alignment for linking the cycle to a calendar.

A periodic event such as “Three times a day for 20 minutes” can be represented by:

```
<effectiveTime xsi:type’:PIVL_TS”>
    <period value = “8” unit = “h”/>
    <phase>
        <width value = “20” unit =”min”/>
    </phase>
<effectiveTime>
```

2.1.2. Temporal Elements

As defined in [9] a temporal element is a finite union of n-dimensional time intervals. There are different kinds of temporal elements defined in the literature. Most frequently used temporal elements are transaction-time elements, valid-time elements and bitemporal elements, which include both transaction and valid time. Addition to these types, user-

defined time elements, availability time elements and event time elements are also introduced. Temporal elements are closed under the set of union, intersection and complementation operations. They can be represented by timestamps or time intervals.

2.2. Temporal Dimensions

In the literature, different kinds of temporal dimensions are defined in order to model time related data [9].

2.2.1. Valid Time

The valid time of a fact is the time when the fact is true in the modeled reality. By associating data with valid time, it is possible to represent not only the current state but also past and future snapshots of the miniworld. Thus, valid times are used to express the lifespan of a given object and the evolution of its properties during this interval. Valid times are usually supplied by the user. A temporal database that supports valid time is called *Historical Database* [9] [10].

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, then the end point of the validity interval is set to the end date in the database.

For example, in a relation that shows the department of each doctor in a hospital; when a doctor starts to work in a department on the 01.12.2006 and if the period of time that he will work is unknown at that time, then the end point of the doctor’s validity interval in that department is set to “Now”. On the other hand, if the validity period is known at the insertion time, the end point of validity interval is set to a specified time. Table 2.1 shows the doctor-department relation records. The first tuple shows that Dr. John works in Intensive Care department between 01.12.2006 and 01.12.2006. The second tuple

shows that Dr. Mary has started working in the Anaesthesia department on 01.06.2007 and she is currently working in that department.

Table 2.1. Doctor-Department Relation

ID	Doctor	Department	Valid Time
1	John	Intensive Care	(01.12.2006,01.12.2007)
2	Mary	Anaesthesia	(01.06.2007,Now)

Temporal databases can support current time in a number of different ways [12]. Maximum approach represents current time as unrealistic large date, which is most often used '31-December-9999'. Another solution to represent current time is to use Null data.

2.2.2. Transaction Time

The definition of transaction time in [9] as follows: "A database fact is stored in a database at some point in time, and after that it is alive until logically deleted". The transaction time of a database fact is the time when the fact is current in the database and may be retrieved. As a consequence, transaction times are generally not time instants, but have duration. Transaction times are consistent with the serialization order of the transactions. They cannot extend into the future and also it is impossible to change the past i.e. past transaction times cannot be changed.

When an object is stored in the database, the end point of its transaction time interval is set to "until changed" (UC). Then when the object is deleted, the system sets the transaction time end point to the deletion date. Hence, deletion is pure logical and the object is not physically removed but ceased to be part of the database's current state. A temporal database that supports transaction time called *Rollback Database*.

Transaction times may be implemented using transaction commit times, and are system-generated and supplied. Differences between transaction and valid times are: while valid times may only be associated with facts, statements that can be true or false,

transaction times may be associated with any database object. Another major difference is that while transaction time is appended only and cannot be updated, valid time can be updated by users.

Valid and transaction times are orthogonal dimensions, i.e. each of them can be independently recorded. However, using only valid or transaction time does not give accurate results for storing and retrieving data.

Bitemporal databases support both valid time and transaction time. While attaching a valid time interval to the object, the database systems record the time when the associated transaction is committed. The database system also sets the end point of the object's transaction time to *UC*. Table 2.2 shows the possible combinations of valid and transaction time dimensions.

Table 2.2. Valid Time-Transaction Time Combinations

Case	Transaction Time Start	Transaction Time End	Valid Time Start	Valid Time End
1	tt1	UC	vt1	vt2
2	tt1	tt2	vt1	vt2
3	tt1	UC	vt1	Now (tt1=vt1)
4	tt1	tt2	vt1	Now(tt1=vt1)
5	tt1	UC	vt1	Now(tt1>vt1)
6	tt1	tt2	vt1	Now(tt1>vt1)

Valid and transaction time can be represented in a two-dimensional space. Figure 2.1 illustrates the possible regions of valid and transaction time combinations. Case 1 shows an alive data record whose valid time points are definite. Case 2 shows a logically deleted record with definite valid time interval. In case 3, valid time end point and transaction time point of the data record is undefined and the record is entered to the database as soon as it is becomes valid. Case 4 shows a logically deleted data record, which is entered to the database as soon as it becomes valid. Case 5 displays an alive record which has been updated retroactively, i.e. TT > VT. In case 6 a logically deleted record is shown which has been updated retroactively.

The below anesthesia scenario will explain the meanings of valid and transaction times in the health domain:

Scenario 1:

On 10.October.2006, the anaesthetist prescribes a bipuvac-based therapy from 15:00 to 19:00 to apply during the surgery. Therapy records are entered into the database at 14:00. However, a complication occurred during the surgery so the bipuvac infusion is stopped at 16:15 and replaced by a diazepam-based therapy from 16:25 to 19:00. The new facts are entered at 17:00. This anaesthesia scenario can be modeled by Table 2.3.

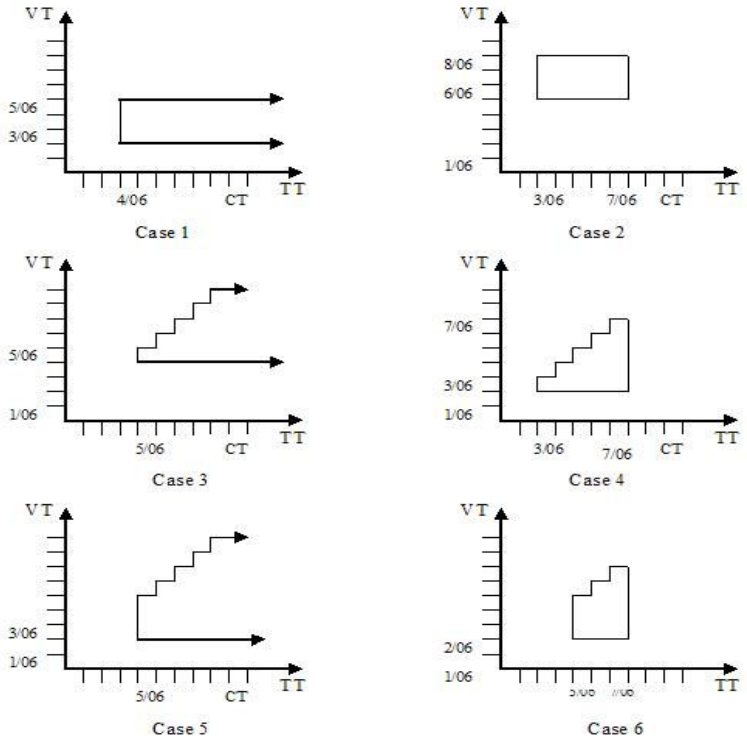
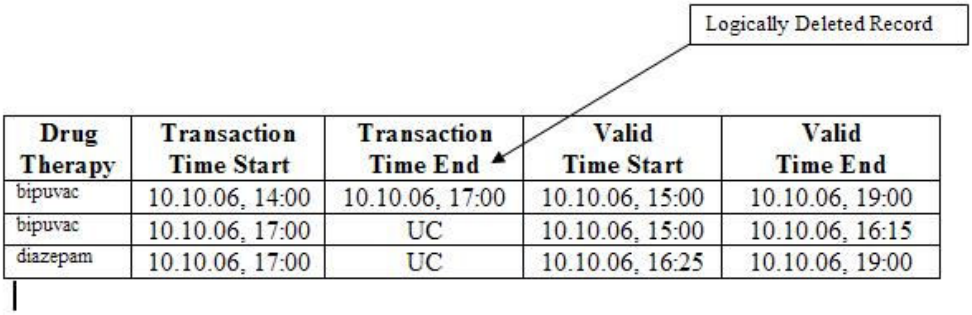


Figure 2.1. Bitemporal Regions

In the above example if the database is a Rollback database, there will be one record for both bipuvac and diazepam. Since the validity intervals of the drugs are not recorded, a query will not answer when the diazepam or bipuvac therapy is applied to the patient. The anaesthetist changed his decision during therapy in the scenario but a rollback database does not show this change. If a historical database is used instead of a rollback database, then there will be two records for bipuvac therapy. When a query is asked to retrieve the valid drug therapy on 10.10.2006 at 16.30, it will return bipuvac (since the records were

entered at 17.00) although the correct answer is diazepam. For this reason in order to retrieve the accurate results, a bitemporal database must be used, i.e. both valid time and transaction time must be included in the data model.

Table 2.3. Anaesthesia Record



Drug Therapy	Transaction Time Start	Transaction Time End	Valid Time Start	Valid Time End
bipuvac	10.10.06, 14:00	10.10.06, 17:00	10.10.06, 15:00	10.10.06, 19:00
bipuvac	10.10.06, 17:00	UC	10.10.06, 15:00	10.10.06, 16:15
diazepam	10.10.06, 17:00	UC	10.10.06, 16:25	10.10.06, 19:00

There exists a variety of possible relationships between valid and transaction times of a given tuple: the starting point of the transaction time can precede the valid time (as in the first tuple of Table 2.3), the starting point of the transaction time can follow the valid time (as in the second tuple), the starting point of the transaction time can follow the starting point of the valid time and precede its ending point (as in the third tuple), and so on. Depending on the modeled application domain, valid and transaction times can be related to each other in different ways.

In the literature several classifications are proposed for valid time and transaction time relationship [9].

2.2.2.1. Retroactive Relation. A temporal relation is retroactive if $VT_S <= TT_S$, where VTs is the start of the validity interval and TT_S is the start of the transaction time interval.

2.2.2.2. Delayed Retroactive With Bound Relation. A temporal relation is retroactive if $VT_S <= TT_S - \Delta t$, where $\Delta t >= 0$.

2.2.2.3. Predictive relation. A temporal relation is retroactive if $TT_S <= VT_S$

2.2.3. Event Time

Valid and transaction times are insufficient to distinguish between retroactive and delayed updates. Another temporal dimension, *Event Time*, is proposed to solve this problem [14]. The 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.

In the literature [15], events are classified as:

2.2.3.1. On-time Events. When an event occurs, its validity starts simultaneously with its occurrence time. For example, a **doctor** can decide a drug administration immediately after a complication in the surgery.

2.2.3.2. Retroactive Events. The validity interval starts before the occurrence time of the event. For example, on October 15, 2006, the manager of the hospital decided an increase of 10 per cent of the salary of the anaesthetists of the Anaesthesia Department, starting from October 1, 2006.

2.2.3.3. Proactive Events. The validity interval starts after the occurrence time of the event. For example on January 29, 2007, the doctor decides a drug administration starting from the February 2, 2007.

The above classification shows the relation between the valid time and event time. Another classification is done on transaction time and event time [15].

2.2.3.4. **On-time update.** The transaction time coincides with the event time. For example as soon as a preoperative test results are available, they are inserted into the database.

2.2.3.5. **Delayed update.** The transaction time is greater than the event time. For example the preoperative test results are inserted some time after their generation.

Event time is different than valid and transaction time in some aspects. Valid time of a fact is an interval since every fact's validity starts at some point in the time line and finishes at another point. Like valid time, transaction time of an object is an interval. Both valid and transaction time right endpoints can be unknown and represented by *"Now"* or *"UC"*.

A fact can be initiated by an event and can be terminated by another event. So the end point of event time of a fact can be null if the fact is not terminated by an event or can be a point if the fact is terminated. Figure 2.2 illustrates the relation of event and valid times.

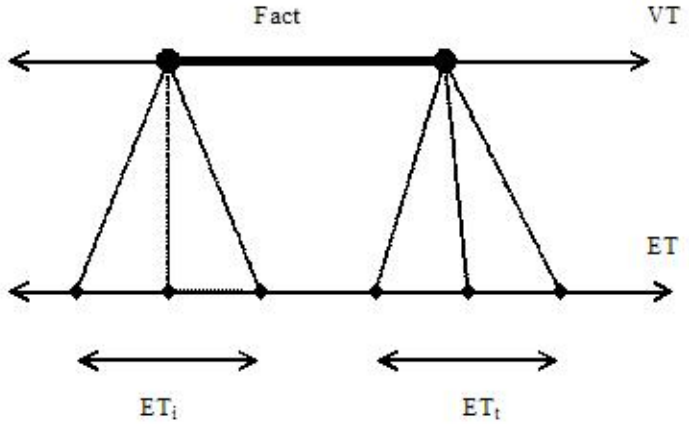


Figure 2.2. Relationship between event and valid times of a fact.

Scenario 2: On October 10, 2006, at 13:00, the physician prescribes a bipuvac-based therapy from 15:00 to 19:00. Data about the therapy is entered into the database at 14:00. Due to the unexpected evolution of the patient state, i.e. a complication, at 16:00 the physician decides a change in the patient therapy. Accordingly, the bipuvac infusion is

stopped at 16:15 and replaced by a diazepam-based therapy from 16:25 to 19:00. The new facts are entered at 17:00.

The anaesthetist decided to apply bipuvac therapy at 13.00 pm. The initiating event of the bipuvac (first record) is the decision of the anaesthetist. When bipuvac infusion is stopped and therapy continued with diazepam because of an unexpected complication, the end point of the validity interval of the bipuvac is set to the 16.15 due to unexpected complication. The terminating event of the bipuvac therapy (second record) is the time of unexpected complication. The anaesthetist decided to continue with a diazepam based therapy instead of bipuvac so the initiating event of the third record is also set to the complication time. Table 2.4 displays the time intervals of the above scenario.

Table 2.4. Records of the patient therapies with valid, transaction and event times.

Drug Therapy	Transaction Time Start	Transaction Time End	Valid Time Start	Valid Time End	Event Time Start	Event Time End
bipuvac	10.10.06, 14:00	10.10.06, 17:00	10.10.06, 15:00	10.10.06, 19:00	10.10.06, 13:00	10.10.06, 13:00
bipuvac	10.10.06, 17:00	UC	10.10.06, 15:00	10.10.06, 16:15	10.10.06, 13:00	10.10.06, 16:00
diazepam	10.10.06, 17:00	UC	10.10.06, 16:25	10.10.06, 19:00	10.10.06, 16:00	10.10.06, 16:00

In the literature, if the terminating event does not exist then it can be represented by *"Null"* or the same time point with the initiating event time [15].

Event times are related to the modeled world and they are used to represent the occurrence times of events (decisions, happenings, actions, etc.) that initiate or terminate meaningful facts of the domain. Figure 2.3 illustrates the relationships between transaction time, valid time and event time.

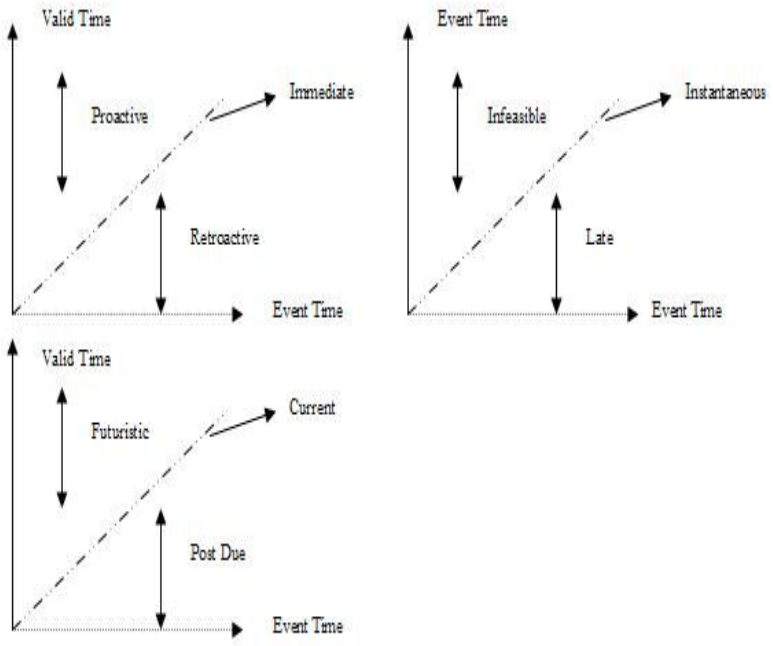


Figure 2.3. Classification of a fact with respect to valid, transaction and event (decision) time

2.2.4. Availability Time

The availability time of a fact is the time interval during which the fact is known and believed correct by the information system [15]. The end point of the interval is the time at which the information system realizes that the fact is not correct. As for transaction time, the end point "UC" means that the fact is currently believed correct.

The information system represents set of information flows of an organization, the human and computer-based resources that manage them. The time that an information system becomes aware of a fact does not always coincide the transaction time or the event time of a fact. The below scenario explains the meaning of the availability time in the health domain.

Scenario 3:

Due to an accident that occurred on September 15, 2006, John suffered from a severe headache starting from October 1. On October 7, John was visited by a physician. On October 9, the physician administered a suitable drug to him. The day after, the physician entered acquired information about John's medical history into the database. On October 15, the patient told the physician that her headache stopped on October 14; the physician entered this data into the database on the same day. Due to an insertion mistake (or to an imprecision in John's talk), the trauma has been registered as happened on September 5, 2006. On October 20, the mistake was discovered. The day after, the physician entered the correct data into the database.
The temporal dimensions in this scenario are:

Event time:
An accident (initiating event in this case) occurred on September 15, 2006.
A drug is administrated to the patient (terminating event for headache in this example) on October 9.

Valid time: A severe headache starting from October 1 to 14 October.

Transaction time:
On October 10 information was entered into the database.
Headache validity interval updated on 15 October.

Table 2.5 shows the valid, transaction, event and availability time dimensions of the data.

Table 2.5. Symptom entity with valid, transaction, event and availability time.

Symptom	T.T Start	T.T End	V.T Start	V.T End	E.T Start	E.T End	A.T Start	A.T End
headache	10.09.06	15.09.06	1.10.06	Now	5.09.06	Null	7.09.06	15.09.06
headache	15.09.06	21.09.06	1.10.06	14.10.06	5.09.06	9.09.06	15.09.06	20.09.06
headache	21.09.06	UC	1.10.06	14.10.06	15.09.06	9.09.06	20.09.06	UC

Using only event, valid and transaction time, it is impossible to express the when the doctor became aware of patient's headache onset and cessation. Availability time of the

first record shows that the doctor has learned the headache symptom on 7.9.2006. The availability time end point of that record is 15.09.2006 since the patient told the doctor that his headache has stopped on that day. In the second tuple the availability start time of the data is same as when the doctor has learned the headache had stopped and the availability time end point is set the day of the discovery of the mistake, i.e. logically deletion of the record.

In many applications, where data insertions are grouped and executed in batches, possibly on the basis of previously filled report forms on paper, transaction time cannot be safely taken as the time when a fact has been acquired by the information system. In some cases, it may happen that the order according to which facts are known by the information system differs from the order in which they are stored into the database. Since there are many application domains, including the medical one, where decisions are taken on the basis of the available data no matter whether or not it is stored in the database, a new temporal dimension, *Availability Time*, has to be introduced, to deal with it [15].

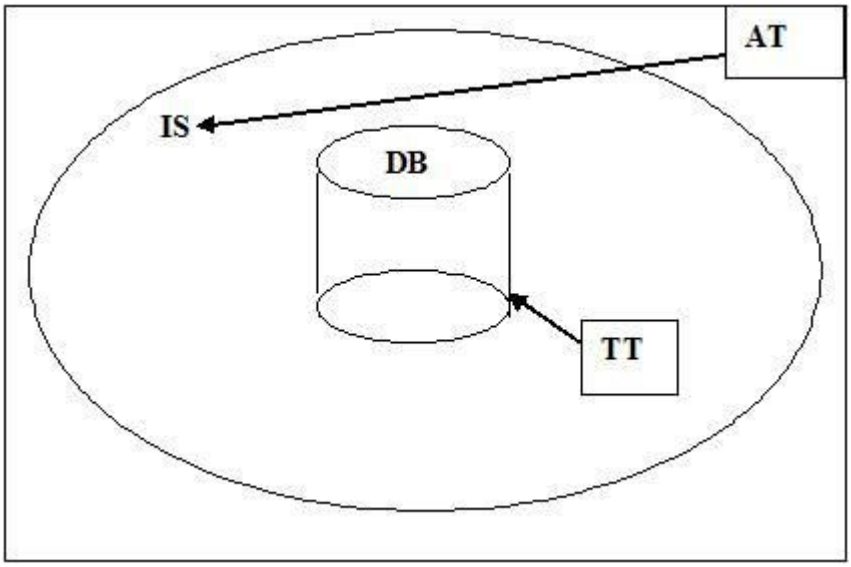


Figure 2.4. Availability (AT) and transaction (TT) times and their relationships with the database (DB) and information systems (IS).

Figure 2.4 illustrates the availability time - information system relationship and transaction time - database relationship. Information system includes, but does not necessarily coincide with the database system.

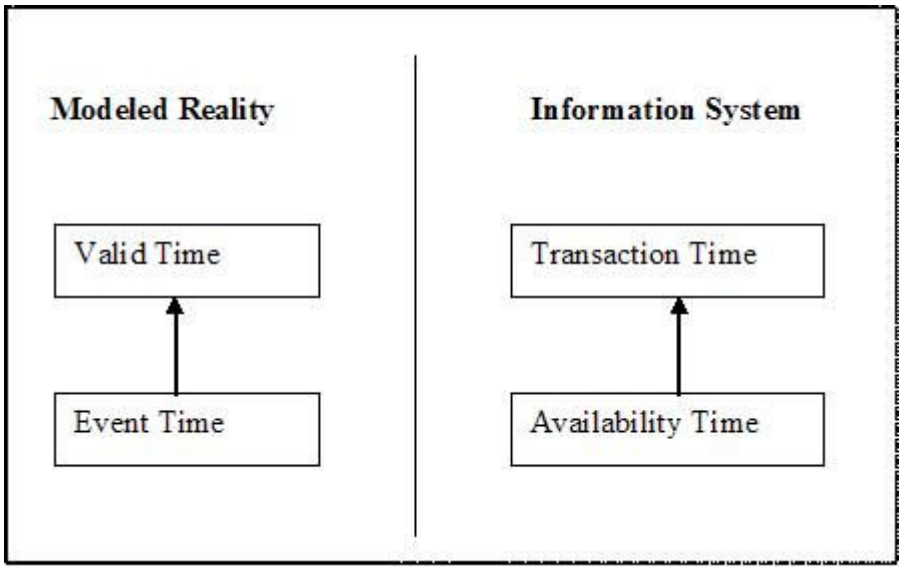


Figure 2.5. Temporal Dimensions, Modeled reality and Information System

Figure 2.5 illustrates the relationships between valid, transaction, event, and availability times, the modeled reality, and the information system. Valid time and event time are meaningful in modeled reality where transaction and availability are meaningful in the information system.

To summarize, transaction time is append-only, i.e. facts previously stored into the database cannot be changed. The acquisition of new knowledge about the domain, results into the addition of new facts, whose transaction time interval includes the current time, to the database; the logical deletion of incorrect knowledge is obtained by "closing" the transaction times of the corresponding facts. In contrast, valid and event times, related to the real world, can be located either in the past or in the future, and they can be modified freely. The availability time is append-only in its nature, because facts previously known and believed correct by the information system cannot be changed. But from the point of database system it is append-only, only if there were no errors in data entry. Consequently, a temporal data model becomes complete with the usage of valid time, transaction time, event time and availability time.

2.3. Temporal Queries

There are different types of temporal queries that are previously defined in the literature [9] [42]:

2.3.1. Temporal Projection

Temporal Projection is pairing the computed facts with their associated times, usually derived from the associated times of the underlying facts.

Ex: Retrieve the blood loss history of patient “P1”.

2.3.2. Temporal Snapshot

Temporal snapshot is extracting a single state of time-varying data at a given instant.

Ex: Who are the patients of the doctor “D1” on 01-01-2006?

2.3.3. Temporal Slicing

Temporal slicing is based on extracting the snapshot of the time-varying documents at a given time point and it consists in computing simultaneously the portion of each state of time-varying data which is contained in a given period and which matches with the structure selection criteria.

Ex: According to the current information what medicines were prescribed to the patient “P1” between 01-01-2006 and 01-02-2006? (This query will take a transaction time snapshot and valid time slice of medicine.)

2.3.4. Temporal Join

Temporal join is basically defined as joining the time-varying data if their intervals intersect. Temporal EquiJoin (TE Join) is joining the time-varying two records if their keys are equal and their intervals intersect. Another different temporal join type is Generalized Temporal EquiJoin (GTE Join), which is basically joining time-varying two records if their keys in a specified range and their intervals intersect a specified interval.
Ex:

There are some complex temporal query types, which use keywords such as: Since and Until.

2.3.5. A Since B

This type of temporal query will extract time-varying data if their time attributes start as the specified time.

Ex: Find the Find the FeO2 values of the patients in X surgery since the drug Y was given.

2.3.6. A Until B

This type of temporal query will extract time-varying data if their time attributes end with the specified time.

Ex: Find the FeO2 values of the patients in X surgery until the drug Y was given.

2.3.7. Continuous Period

This type of query will extract time-varying data if their interval is matches with the specified time period.

Ex: Find the patients who have been administrated the drug X for more than 3 hours in surgeries with general anaesthesia.

2.3.8. Period Containment

Period Containment is extracting the time-varying data records if their time intervals contains the specified period.

Ex: Find all the patients with the same surgery history as the patient “P1”.

2.3.9. Changed Detection

Change Detection is basically identifying updated time-varying data.

Ex: Find all updates of the patient anaesthesia drug therapies that were applied retroactively. (i.e. TT > VT)

2.3.10. When

This type of query is used for applying temporal constraints on time dimensions.

Ex: Find all surgeries that appeared no later than 30 minutes after their initiating event.

2.4. Temporal XML Data Models and Native XML Database Systems

Temporal databases have been an active research topic for twenty years. Most of the applications are temporal in nature so users’ demand for temporal database applications is increasing with time. However, database vendors are not moving forward in supporting the management and querying of temporal information. In the last years, some of the vendors take this demand into consideration in their commercial products. These products provide

support in various ways for transaction and valid time data. Some of the commercial databases, which give temporal support, are listed below [25]. However, all these database products support temporal data in relational data models.

TIMEDB [26] is a temporal relational database management system. It offers a temporal query language interface to valid-time, transaction-time, and bitemporal tables. The temporal query language is temporally upward compatible with SQL.

FlashBack queries in Oracle 9i and 10g support temporal management [27] [28]. Oracle 9i FlashBack allows transaction timeslice queries. Oracle 10g extends the flashback queries to retrieve all the versions of a row between two transaction times, i.e. it allows key-transaction-time-range queries.

LogExplorer [29] from Lumigent provides an analysis tool for Microsoft SQLServer logs, to allow one to view how rows change over time and then to selectively back out and replay changes, on both relational data and the schema.

In the last years XML has been widely used in researches for temporal data models. XML hierarchical structure can naturally represent the history of databases in a temporally grouped data model and powerful temporal queries can be expressed in XML based query languages without requiring any extension to current standards. Because of this reason, XML data models are more convenient than relational data models for temporal data management.

XML documents can be divided into two categories: *Data Centric* and *Document Centric*. In data-centric documents XML is used as a data transport. Examples of data centric documents are: sales orders, patient records, and scientific data. In document-centric documents are XML is used for its SGML-like capabilities. Examples of document centric documents are: user's manuals, static Web pages, and marketing brochures. There are two basic choices to store XML documents: a native XML database or an XML-enabled database.

XML-enabled databases hold data in some format other than XML. An interface is provided so that XML can be presented to an application even though the data is stored in some other format. An XML-enabled database might be a relational database, object-relational database, or an object-oriented database.

Native XML databases allow XML data to be stored directly. They are likely to perform better than XML-enabled databases since there is little need for converting the data or that the conversion is minor. The data conversion in an enabled database is almost always going to be more significant and time consuming than with a native database.

Temporal management is much easier in XML databases than relational databases. Because of this, native XML databases are convenient for temporal data models. Some of the native XML databases are listed below:

eXist [30] is a native XML database that uses a proprietary data store which contains B+ trees and paged files. Documents are stored in a hierarchy of collections and collections can contain child collections. The XML data store represents the central component of eXist's native storage architecture. The documents are stored according to the W3C's document object model. The DOM implementation completely relies on the numbering scheme to determine node relationships. eXist uses four index files: "Collection.dbx" manages collection hierarchy, "Dom.dbx" nodes in a paged file and assigns unique node identifiers to actual nodes, "Elements.dbx" indexes elements and attributes and "Words.dbx" tracks of word occurrences, for full text search extensions. These indexes based on B+ trees not by document but collection.

ENAXS [51] is another native XML database, which focuses storing nodes of the same path types together in a group. It reduces storage space and provides faster data retrieval as most queries expect data with a same structural attribute from multiple documents. ENAXS deals with the approach that all root nodes, internal nodes (elements) and external nodes (contents) from multiple documents are collectively grouped according to their paths, and stored together in the repository. Main components of the ENAXS are: Node repository which maintains all root and internal nodes, value repository that contains binary form of all external nodes, node group which is a set of node-references grouped

together according to their paths regardless of which document they belong, a schema tree that is an abstract representation of the structure of XML documents and a path index which maintains a set of keys which are hash values generated from the path expressions. The leaf nodes of the index tree contain references pointing to the associated node groups.

Xindice [31] is another native XML database written in Java, which is designed to store large numbers of small XML documents. It stores and indexes compressed XML documents to save space. Data is arranged into a hierarchy of collections and can be queried with XPath. There are two important layers in Xindice: Paging layer and B+ tree layer in which key and value pairs are optimized for disk accesses. For every XML document, Xindice calculate "compressed DOM". In the B+ tree layer, xml documents are stored as (name of the document and compressed DOM value) tuple.

Sedna [32] is an open source full featured native XML database system that supports queries, updates and security. It is based on the XQuery language and the XQuery/XPath data model. It provides efficient support for unlimited volumes of document-centric and data-centric XML documents that may have a complex and irregular structure. Sedna uses direct pointers to represent relations between nodes of an XML document such as parent, child and sibling relationships.

Natix [33] is a native XML database designed to support compact storage of structure and content of XML documents, and its index structures for content and structure retrieval. Natix accesses the raw disks or file system files and provides a memory space divided into segments. A large XML tree is divided into subtrees and stored into one disk page.

Timber [34] is a native XML database that has architecture as close as possible to that of a relational database. The Timber has an XML algebra that manipulates sets of ordered, labeled trees. It supports a number of different types of indexes, including element, attribute, and text, inverted, parent, and join indexes. Timber supports a subset of XQuery.

The XML databases that are mentioned in this subsection are some of the native XML database products. However, none of them support temporal data management. Temporal dimensions can be added to these databases as user defined times. Using user

defined times have some disadvantages such as users have to define all temporal attributes beforehand and are responsible for maintaining the relation keys.

In order to support temporal dimensions in native XML databases, temporal XML data models are needed. In the literature there are some researches for temporal XML data models. Much research work has recently focused on adding temporal features to XML, which helps to support change, versioning, and evolution of XML documents. Some of the temporal XML based data models are listed below:

XBIT - An XML Based Bitemporal Data Model [35]

XBIT logical data model is basically showed that valid-time, transaction-time, and bitemporal databases can be naturally viewed in XML using temporally-grouped data models which are compatible with the hierarchical structure of the XML. Temporally grouped data models have some advantages over temporally ungrouped data models such as: no redundant information is preserved between tuples and there is no need to coalesce frequently tuples by temporal queries. Table 2.6 and 2.7 show the temporally grouped and ungrouped tables.

In temporally grouped data models the time stamped history of each attribute is grouped under the attribute.

Table 2.6. Bitemporal History of patients in a temporally ungrouped data model

Patient Name	Doctor Name	Drug Name	Valid Time	Transaction Time
John Brown	Dr.Jim Williams	Bipuvac	(13:00, 15:00)	(13:00, 14:00)
John Brown	Dr.Jim Williams	Bipuvac	(13:00, 14:00)	(14:00, UC)
John Brown	Dr.Jim Williams	Diazepam	(14:00, 15:00)	(14:00, UC)

Table 2.7. Bitemporal History of patients in a temporally grouped data model

Patient Name	Doctor Name	Drug Name
John Brown V: (13:00, Now) T: (13:00, UC)	Dr.Jim Williams V: (13:00, Now) T: (13:00, UC)	Bipuvac V: (13:00, 15:00) T: (13:00, 14:00)
		Bipuvac V: (13:00, 14:00) T: (14:00, UC)
		Diazepam V: (14:00, 15:00) T: (14:00, UC)

With temporal grouping, the bitemporal history is represented as an XML document (BH-document). Each element in the BH-document is assigned two pairs of attributes tstart and tend to represent the transaction time interval; and vstart and vend to represent the valid time interval. There is a covering constraint whereby the transaction time interval of a parent node must always cover that of its child nodes, and likewise for valid time intervals. In figure 2.6 the BH document of patient relation is displayed. Some of the dates are not shown for the simplicity.

The XBIT data model support powerful temporal queries, expressed in XQuery without requiring the introduction of new constructs in the language.

Although 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 structure, which supports efficient temporal management, does not exist in the model.

```
<patient vstart=01.01. 2006  vend=now tstart=01.01. 2006  tend=UC >
<patienName vstart=01.01.2006  vend=now tstart=01.01. 2006  tend=UC>
        John Brown
</ patienName >
<doctorName vstart=13:00 vend=now tstart=13:00 tend=UC>
        Dr.  Jim Williams
</doctorName>
<drugName vstart=13:00  vend=15:00 tstart=13:00   tend=14:00  >
        Bipuvac
</ drugName >
<drugName vstart= 13:00  vend=14:00  tstart=14:00   tend=UC>
        Bipuvac
</ drugName >
<drugName vstart=14:00   vend=15:00  tstart= 14:00    tend=UC>
        Diazepam
</ drugName >
</patient>
```

Figure 2.6. BH document of patients

Adding Valid Time to XPath [24]

In this data model, XPath query language and data model is extended to include valid time. The primary contributions of the paper are: a valid-time view, which a valid time is presented, is proposed. The view is a calendar- and query-specific rendering of the valid time as a virtual XML document. Second, XPath is extended with a valid-time axis. The role of the axis is to provide users with a query language mechanism for accessing valid times and to keep wild-card queries from exploring the valid time.

The proposed valid time XPath data model D_{VT} for a well-formed XML document X is a four-tuple D_{VT} (X) = (r; V;E; I) where V is the node set, E is the edge list, I is the information set and r is the root. The time attributes, which are either disjoint intervals or instants, are attached to nodes. Every node of the tree structure of a well-formed XML document is associated with the valid time that represents when the node is valid, no node can exist at a valid time when its parent node is not valid. The valid time of any node is a superset of the union of the valid times of all its children as well as all its descendents. The

valid time of the root node should be a superset of the union of the valid times of all nodes in the document. The valid time of an edge is determined by the valid time of the nodes at the edge's two ends. The valid time of the edge is result of $t_1 \cap t_2$, where t_1 and t_2 are the valid times of the edge's two ends.

The XPath is extended in the paper with an axis to locate the valid time of a node. Each node in an XML document has a corresponding valid time view containing its valid time information. A valid time axis is added to the query language to retrieve nodes in a view of the valid time for a node. The valid time axis selects the list of nodes that forms a document-order traversal of the valid time view.

Some examples, which use the valid time axis to query within the default view of the valid time, are given in Figure 2.7.

v/valid	specifies the valid time axis of the node v.
v/valid::day	selects all the day nodes in the axis.
v/valid::time[2]	selects the second time node in the axis .
v/valid(``Gregorian")	specifies that the calendar to use in the valid time axis of v is \Gregorian".

Figure 2.7. Sample queries

In this model only valid time dimension is supported in an extension to XPath data model. However, the paper does not propose a storage structure or an index structure in order to store and retrieve XML data efficiently.

A Data Model for Temporal XML Documents [36]

The contents of XML documents may change as time goes by, so that a temporal data model is needed. A logical data model based on XPath for temporal XML documents is proposed to represent history of temporal documents. The extensions of the model are: edges have a label that represent their valid time, string-value of text and attribute nodes are modeled as virtual nodes and text and attribute nodes can contain multiple string-value nodes. One temporal dimension, valid time, is used in the model.

XPath data model is extended to record the history of documents in the model. Some physical implementation methods are also proposed. There are two methods; one is for implementing the data model to XML documents as they are (full implementation), and the other is for implementing the data model retaining the original form of XML documents (simplified implementation).

In full implementation, every node and valid time labels are mapped into separated elements and attributes. Three items are introduced for this reason: valid attribute, attribute node, and string-value node. A valid time label is mapped to a valid attribute. Thus, every element has one valid attribute. An attribute node is mapped to an attribute element since attributes may have multiple string-value nodes over time. In full implementation multiple stringvalue elements are included, in an attribute element. A string-value node is mapped to an element stringvalue in the implementation.

Although full implementation captures all information of the data model, some of the data is redundant and it makes the document different from its original document. A simplified implementation is introduced for this reason. In the simplified implementation, only valid attributes are used, and represent the history of a document by duplicating elements, which are updated.

There are no specialized techniques for querying temporal documents. A snapshot of database is obtained and then the snapshot is queried. There are no indexing structures for querying data and only valid time is supported in the model.

Temporal modelling and management of normative documents in XML format [37]

The model is based on a hierarchical organization of normative texts. There are four different temporal dimensions in the model in order to represent the evolution of norms in time and their resulting versioning correctly.

Publication time is the time of publication of norms in an official journal. Valid time represents the time the norm is in force. Efficacy time usually corresponds to the validity of norms, but sometimes the cancelled norm continues to be applicable to a limited number

of cases. Finally, transaction time is used as the time when the norms are stored in the computer system. All the temporal dimensions except the publication time are independent of each other so that they are orthogonal. Publication time is a global property of the document, which cannot be changed after publication and, it is not involved in the versioning mechanism.

The model represents the norms in an XML based data model, which is also enriched with timestamping to make versioning possible. A simple part of the XML Schema of the legacy documents is displayed in the figure 2.8 and figure 2.9.

```
<norm num=10 type=law publication=01.12.2006 id=001102>
<title>Cereals importation</title>
<contents vts=05.12.2006 ets=05.12.2006 tts=05.12.2006>
<ver num=1>
<section num=1>
<ver= num=1 <heading> import from commutarian countries</heading>
<article num=1>
<ver= num=1 <heading> import from spain</heading>
.....
</article> </ver> </section> </ver> </contents> </norm>
</norm>
```

Figure 2.8. Sample xml document for a norm.

The proposed model is different from the other temporal models since it supports multiple temporal dimensions. The model is implemented on an XML-enabled system and design. There are not any storage structures or indexing technique specialized for temporal dimensions mentioned in the model.

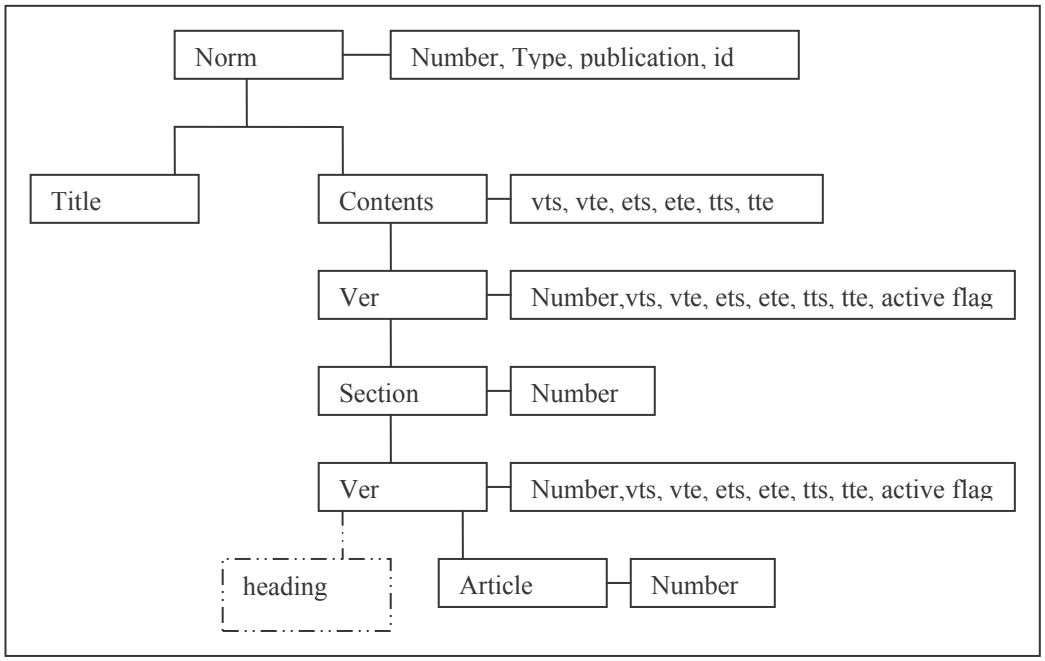


Figure 2.9. Sample xml schema document for a norm.

Temporal XML: Model, Language and Implementation [40]

A temporal XML data model is proposed for tracking historical information in an XML document, and for recovering the state of the document at any given time. Figure 2.10 displays the proposed data model graph. Valid time is supported by the model, but the authors claim that transaction time can be also applied in the data model as valid time. XML documents are represented by a directed labeled graph that contains containment edges for describing element nesting and attribute values, and reference edges for IDREF to ID references. Both kinds of edges may be labeled with temporal elements; so that at any given instant when a snapshot of the temporal document is extracted the result is a conventional XML document. There are four kinds of nodes in the model: Root node represents the root element in the XML document such that root node has no incoming edges, and every node in the graph is reachable from the root node. Value nodes represent the text or numeric values in the XML document. They have no outgoing edge and one incoming edge from the attribute or element nodes. Attributes nodes are labeled with attributes names and possibly one of the ‘ID’ or ‘REF’ annotations. Finally element nodes are labeled with element tags and contain outgoing edges to attribute nodes, value nodes or other element nodes. In the model, an element node can have more than one value node.

Each node is uniquely identified by an integer, *node number*, and described by a node label. An edge connects two nodes if and only if node 1 is either root or an element node and node 2 is an attribute or value or an element node; or if node 1 is an attribute node then node 2 is a value node. A reference edge links an attribute node, node1 of type REF, with an element node, node2.

If an element contains another element than containment edge is constructed between two nodes in the model. Edges are labeled with timestamps. For example, if an element is valid between time1 and time2 then the edge from its parent to the element is labeled with the interval (time1, time2). This means that the element is valid in the interval (time1, time2) and the edge is alive in that interval.

Version nodes are introduced in the model to represent the historical changes in the elements. A versioned node is a pair consisting of a node in the graph and an ordered list of k element nodes or attribute nodes of type other than ID or REF. Time labels of containment edges of the version nodes are consecutive, i.e. TimeLabel(edge i). To =TimeLabel(edge i+1).From -1.

Four different implementation techniques are proposed for the graph implementation: Top-Down Non-replicated implementation, Bottom-Up Non-replicated, Node replicating and Node-Edge representation. However, the non-replicating representations have much lower space requirements than the others, without a large increase in snapshot generation time.

A temporal XPath query language, which extends XPath with temporal operators, is presented.

In temporal XPath, the meaning is a sequence of node, interval pairs such that the node has been continuously at the end of a matching path during that interval. Maximal continuous path is defined for this reason. Continuous path is path with interval T from node 1 to node n if there is a sequence of edges of the form e_1 (node1, node2, T_1) …… e_n (node n-1, node n, T_n) where $T= \cap_{I = 1, n} T_i$. Maximum continuous path is a continuous path

with interval T from node 1 to node n if T is the union of a maximal set of consecutive intervals T_i such that there is a continuous path from node 1 to node n with interval T_i.

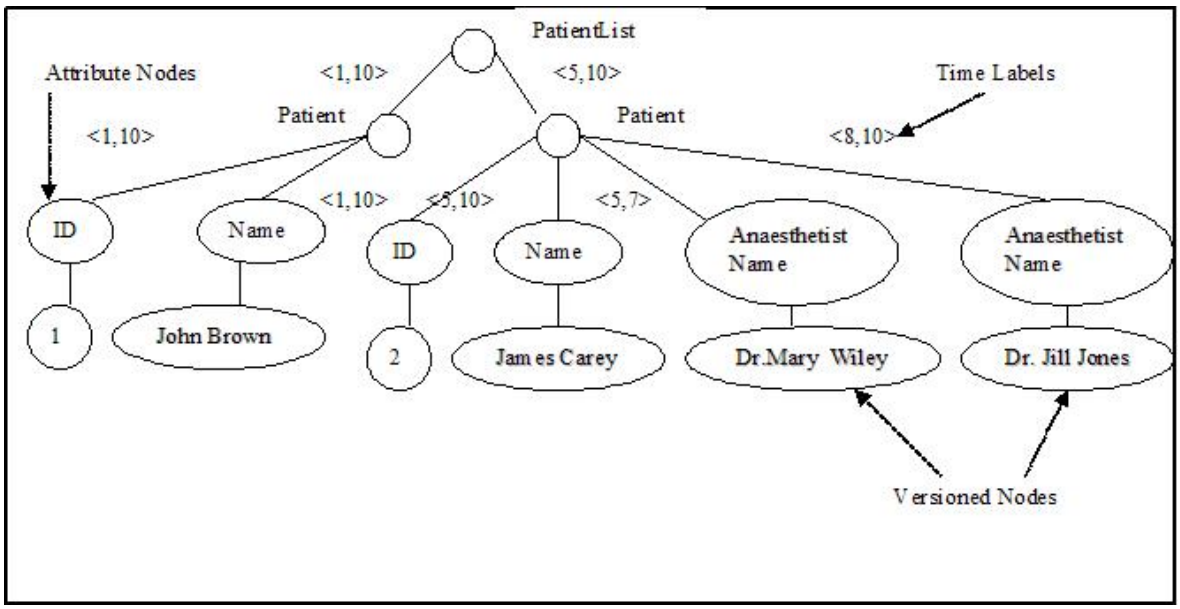


Figure 2.10. Sample graph of the data model.

There are some other temporal data models that are designed to support version management or change management of XML documents. Diff based models [38] [39] stores the latest version together with all forward completed changes between successive versions. In these models, a record of changes, delta, is kept between every pairs of consecutive versions. Diff algorithms are used to compute deltas and the latest version is stored together with all forward completed deltas, i.e. changes between successive versions that can allow one to get to an earlier version by inverting deltas on the latest version. The data models designed for version management of XML documents are based on the sequence of snapshots and the results of difference algorithms between consecutive snapshots.

2.5. Multidimensional Index Structures

Temporal database systems support efficient storage and retrieval of time-related data with the help of data structures, which are specially designed for the time dimensions.

When the number of time dimensions in the data model is more than one, then more complex data structures have to be used for efficiency.

Existing research shows that regular indices such as B-trees are not suitable for temporal data. The majority of the proposed temporal index structures are for transaction-time, and only few support valid-times [44]. Even less research has been done on the indices for bitemporal data [45]. Due to the similarities between temporal and spatial data in terms of multiple dimensions, spatial indices can be adapted for indexing temporal data. For this reason, multi-dimensional index structures, which are used for spatial data indexing, are also good candidates for indexing temporal dimensions because of the similarity of their multi-dimensionality property.

In the literature, index structures such as R tree, R* tree, Multiversion B tree, Bitemporal R Tree, Bitemporal Interval tree and Adaptive R* tree are proposed for multi-dimensional data [46] [47] [50].

2.5.1. R Tree

An R-tree is defined in the literature as [46], a height-balanced tree similar to a B-tree with index records in its leaf nodes containing pointers to data objects. Nodes correspond to disk pages if the index is disk-resident, and the structure is designed so that a multidimensional search requires visiting only a small number of nodes. R tree stores multidimensional rectangles as complete objects without transforming its dimensions. The index is completely dynamic; insertions and deletions can be intermixed with searches and no periodic reorganization is required. The R tree allows overlapping rectangles so it cannot guarantee that only one search path is required for an exact match query.

The structure of R Tree can be described as follows: Leaf nodes of the R-tree consist of a unique identifier of the data object and a *k*-dimensional bounding box which bounds its data objects. Non-leaf nodes contain a child-pointer pointing to a lower level node in the R-tree and a bounding box covering all the boxes in the lower level nodes in the sub tree. Figure 2.11 illustrates the planar representation and the structure of an R-tree. To insert an

object, the tree is traversed and all the boxes in the current non-leaf node are examined. The constraint of least coverage is employed to insert an object, i.e. the box that needs least enlargement to enclose the new object is selected. Ties are solved by choosing the smallest area. The nodes in the sub tree indexed by the selected entry are examined recursively. Once a leaf node is reached, a straightforward insertion is made if the leaf node is not full. However, if the leaf node is full then splitting is needed before the insertion. For each node that is traversed, the covering box in the corresponding parent node is re-adjusted in order to bind the entries tightly in the node. For a newly splitted node, an entry with a covering box, which is just large enough to cover all the entries in the new node, is inserted into the parent node if there is sufficient space in the parent node. Otherwise, the parent node is splitted and the process may propagate to the root. To remove an object, the tree is traversed and each entry of a non-leaf node is checked to determine whether the object overlaps its covering box. For each such entry, the entries in the child node are examined recursively. The deletion of an object may cause the leaf node to underflow. In this case, the node needs to be deleted and all the remaining entries of that node should be re-inserted from the root. The deletion of an entry may also cause further deletion of nodes in the upper levels. Thus, entries belonging to a deleted i^{th} level node must be re-inserted into to the same level of the tree. Deletion of an object may change the bounding box of entries in the ancestor nodes. Hence, re-adjustment of these entries is required. In order to locate all the objects, which intersect a given query box, the search algorithm descends the tree starting from the root. The algorithm recursively traverses the sub-trees of those bounding boxes that intersect the query box. When a leaf node is reached, bounding boxes are tested against the query box and their objects are fetched for testing whether they intersect the query box.

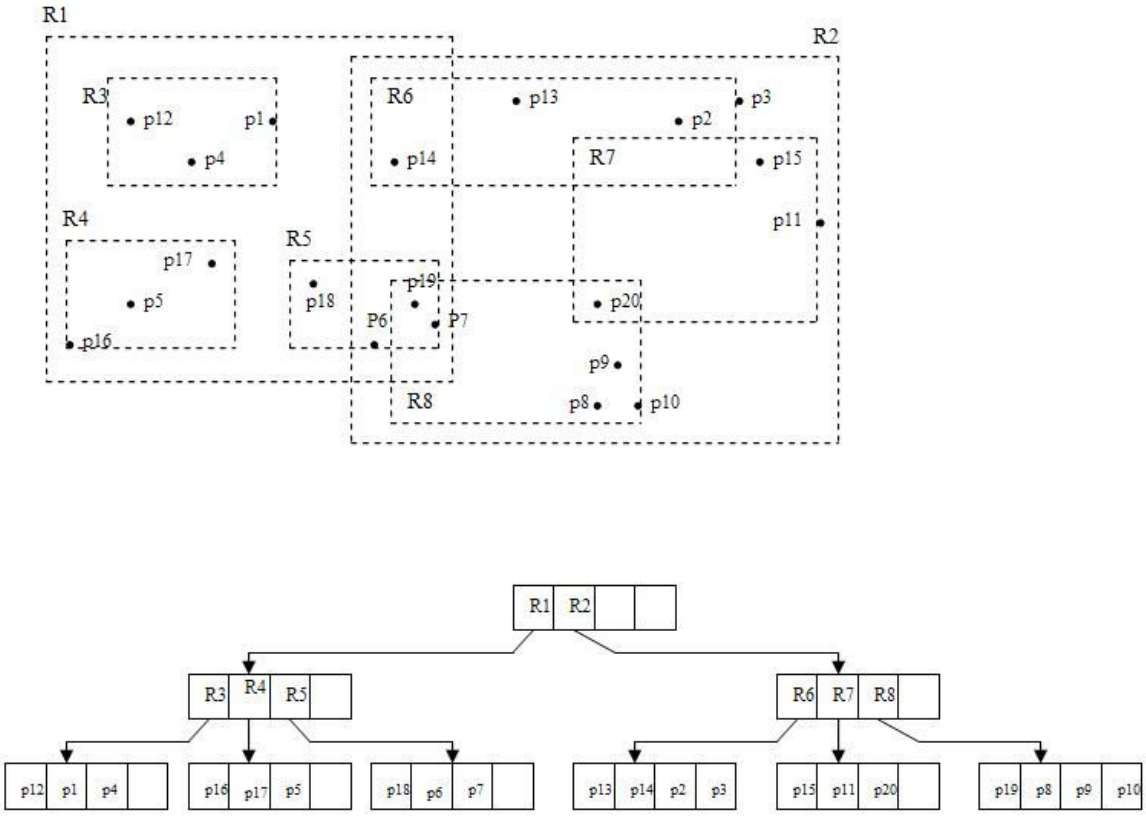


Figure 2.11. The planar representation and the structure of an R-tree

2.5.2. R* Tree

R* Tree is one of the basic index structure that is frequently used in multidimensional indices. It is defined in [47] as an optimized variant of R Tree. Structure of the R* tree is similar to R tree but the heuristics that are used during operation are optimized. The ChooseSubtree algorithm, which is used during an insertion, places a new entry in the tree. It starts at the root node and traverses the tree. At each visited node, the algorithm places a new entry in the sub tree where the placement of the entry leads to the least enlargement of the overlap between the bounding regions of the sub-trees of the node. To determine the overlap enlargement when placing an entry in a sub tree, the overlap between the sub-tree's minimum bounding region, not including the new entry, and the minimum bounding regions of all the other sub-trees is determined. Then the overlap, when the minimum-bounding region of the sub tree is extended with the new entry, is determined, and the overlap enlargement resulting from the placement of the entry is found. The sub tree, or

node, where including the new entry yields the least overlap-area enlargement, is selected. Ties are resolved by choosing the node whose minimum bounding region requires the least area enlargement when including the new entry, and further ties are resolved by choosing the node whose minimum bounding region has the smallest area with the new entry enclosed. The R*-tree Split algorithm investigates a subset of all the possible distributions of entries into two nodes and finds the best distribution according to three heuristics: Firstly, the sum of the margins of the resulting bounding rectangles (margin-value of the distribution) should be as small as possible. Secondly, the overlap between the resulting bounding rectangles (overlap-value of the distribution) should be as small as possible. And finally, the sum of the areas of the resulting bounding rectangles (area-value of the distribution) should be as small as possible.

2.5.3. Double R Tree

Although R* tree can index time dimensions, it has a major disadvantage of overlapping and dead space which results from maximum timestamp approach. When many intervals end at *now*, i.e. for transaction time when data is alive, to keep the now-relative data in a separate structure is a better solution. Double-tree methodology as mentioned in [50] avoids overlapping problem while retaining the advantage of using off-the-shelf access methods. Although it is implemented by two R*-trees various other multidimensional access methods could be facilitated. When an object with valid-time interval I is inserted in the database at transaction-time t, it is inserted at the front R-tree. The front R-tree keeps alive objects for which the right 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 as a rectangle of height I and width from t to t_e on the back R-tree. The back R-tree keeps logically deleted objects with known transaction-time interval. The temporal information of every such object is thus represented simply by a valid time vertical interval that "cuts" the transaction axis at the transaction-time this object was inserted in the database. Insertions in the front R-tree objects are in increasing transaction time while logical deletions can happen anywhere on the transaction axis.

To answer a point valid time (v_j) and a point transaction time (t_i) query, the back R-tree is searched for all rectangles that contain point (t_i, v_j). The front R-tree is searched for all vertical intervals which intersect a horizontal interval H. Interval H starts from the beginning of transaction time and extends until point ti at height v_j. The front R tree indexes start of the transaction time so a point on transaction time means that data is alive from the point until now. By searching the transaction time points from beginning of transaction time to t_i means that it searches all current data whose transaction start time is between beginning of transaction time and t_i. Figure 2.12 illustrates the example query on the Front and Back R tree.

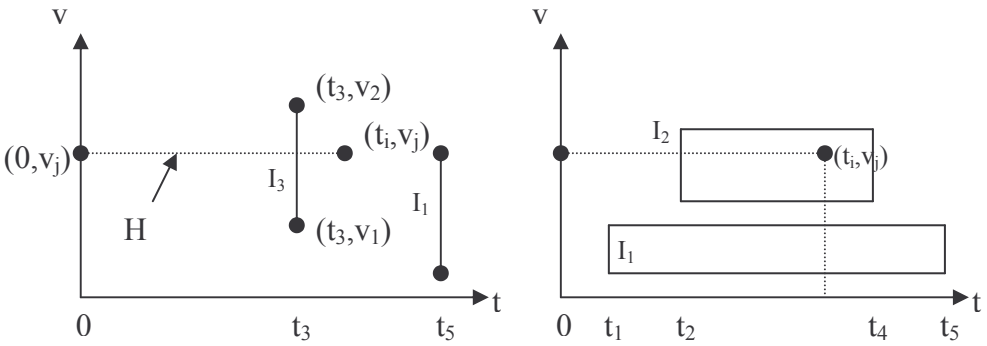


Figure 2.12. Front R Tree (left), Back R Tree (right).

2.5.4. AR* Tree

As mentioned in [49], the multidimensional index structures, such as R*-Tree [47], X-Tree, Kd-Tree [48], are designed for all-dimensional range queries (AD queries) in which a query range is given in each of index dimensions. If these indices are used for partially dimensional range queries (PD queries), then the information in the irrelevant dimensions also has to be read from disk. AR* tree is proposed in [49] 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 in an n-dimensional space, the range queries may use d dimensions of the n dimensions where d is smaller than

n. For example, the range query with d_1 and d_2 as query dimensions is a PD range query for the four-dimensional index built with d_1, d_2, d_3, and d_4 as index dimensions.

Using one n-dimensional index in the entire n-dimensional index space, one PD range query with d (d < n) query dimensions can be evaluated by simply extending the query range in each of the (n − d) irrelevant index dimensions to the entire data domain. Multiple indices can also be used for multidimensional data. If the data is in n-dimensional space, then one index structure is built for each dimension. For example, Multi-B trees can be used for multidimensional data indexing. In this approach, one B-tree, or a variant of it, is constructed in each index dimension, using the projections of the objects.

For PD range queries, the corresponding B-trees are used individually and their results are intersected to obtain the final query result. Figure 2.13 illustrates the result of a partially dimensional range query on multiple indices. The bold shadow region is the result of the query range. Two range queries are firstly evaluated on the two corresponding B trees. All of the data in the A, B, C and D regions are intermediate results, R1 and R2, and then the final result of this PD range query is given by R1 ∩ R2.

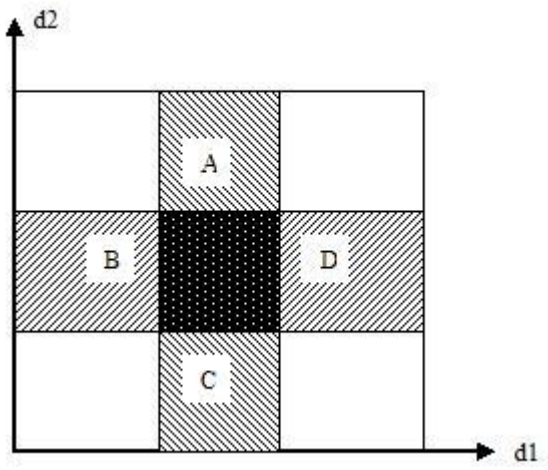


Figure 2.13. Sample graph of the data model

The key concept of AR* tree as mentioned in [50] is to divide each of the n-dimensional R* tree nodes into n one-dimensional nodes, form a node-group, each of

which holds the information in one dimension, while each node of R*-tree holds the information in all of the index dimensions. Every entry in R* tree nodes includes the MBR information in all of the dimensions, each entry in the nodes of AR* tree includes only one-dimensional information. In each node-group, each set of entries having the same index (location) and distributed in different nodes forms an entry of node-group, which corresponds to a complete MBR. Each entry of every node in one node group corresponds to an edge of the MBR, while each of the entries in the index nodes of R*-tree corresponds to a complete MBR. Figure 2.14 illustrates the structure of AR* tree.

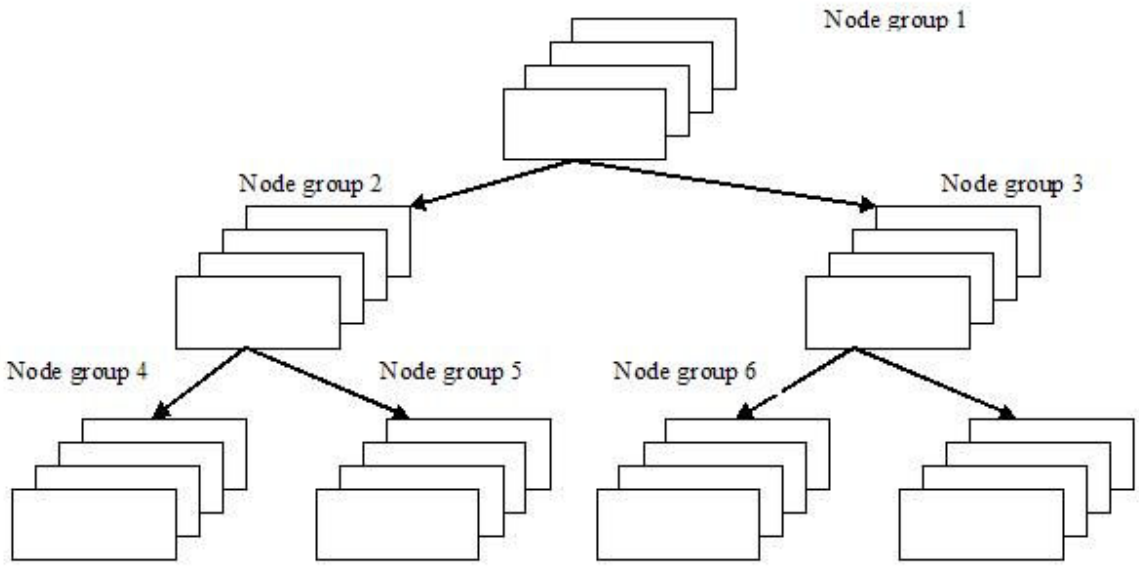


Figure 2.14. Structure of AR* Tree

The structure of AR* tree guarantees that it can be applied to PD range queries with any combinations of query dimensions and that only the relevant one-dimensional nodes are visited. Insertion and deletion algorithms are naïve extensions of original R* tree algorithms. An algorithm for range query search is shown below.

Range Query Search Algorithm:

Input: rect: query range, node-group: initial node-group of the query
Output: result: all the tuples in rect
Begin

For each entry e in node-group Do
If e INTERSECT rect in all the query dimensions) Then
If (node-group is not at leaf) Then
RangeQuerySearch (rect, e.child); //e.child means the child node-group of e
Else result ← e
EndFor
End

An entry includes all the parts with the same index in the nodes of this node-group. In the visited node-groups, not all nodes of the query dimensions are necessary to be checked.

Starting with the root node-group, each entry of the current node-group needs to be checked as to whether its MBR intersect the query range. If its MBR intersects the query range, and the current node-group is not at the leaf level, then this algorithm is invoked recursively with the corresponding child node-group. When each entry e of the current node-group is checked, not all of the nodes in this node-group have to be accessed (irrelevant nodes are skipped), and even, not all of the nodes in the relevant dimensions (query dimensions) have to be visited, because further checks are not necessary after the current entry is found not to intersect the query range in the current dimension.

As stated in [50], Adaptive R* tree has a clearly better performance for PD range queries than the naive methods, R*-tree and multi-Btree. Although R* tree was used in proposal, the other hierarchical MBR-based multidimensional indices can also be employed.

2.6. An Information System in Health Domain: Anaesthesia Information Systems

Time-oriented applications are used in most of the clinical areas. Anesthesiology is one of the important areas in health domain. Systematic structures for the anaesthesia documentation systems have been using for a long time. A properly prepared record, accurately documenting the information of anaesthesia is important not only to the anesthesiologists in detecting untoward trends during the case but also is important for the post operative care of the patients. It is also main source of the information for those who are responsible for providing anaesthetics to the patient. Anaesthetic records are very important for education and research, and are obliged by most of the countries because of

medico legal reasons. In court of law, the medical record is obligatory and considered as the primary source of facts. Anaesthesia information consists of: pre-operation assessment, lab tests and medication, anaesthesia log, operation summary and post-operation status information that are obtained from pre-operative, intra-operative and post-operative phases of anaesthesia.

In the past, paper records were the basic structure of the anesthesia documentation systems. However, using paper-based records bring some disadvantages. According to a number of investigations, 20 per cent of the total anaesthesia time is required for documentation so that doctors prefer to fill the anaesthesia documents after the surgery from their memory. This yields anaesthesia documents to be less accurate. As the time progress the introduction of electronic anaesthesia systems becomes inevitable due to public awareness and medico legal aspects.

Basically, an Anaesthesia Information Management System (AIMS) captures anaesthesia related information in a digital format [43]. An essential component is the Automated Anaesthesia Record (AAR) keeper that collects information about intra-operative activities. However, an AAR does not qualify as an AIMS as it does not allow the information to be used for the management purposes. AIMS advantages include not only speed and accuracy but also research and practice analysis. Another advantage is that with AIMS, it is easily possible to integrate with health information systems such as ADT (Admission, Discharge, and Transfer) system, an operating room management system, laboratory data retrieval system, and drug supply, patient billing, and accounting system. This integration reveals the need of easy exchange of documents and a common structure for AIMS. In the literature it is indicated that [43], the next generation of AIMS systems will need a schema to define the structure for well-formed XML anesthesia documents.

An XML Schema for anaesthesia information systems is proposed in 2002 [7] in order to provide a common structure for anaesthesia data. Currently, a HL7 Special Interest Group, which was formed in 2005 (SIGGAS), works towards creating HL7 CDA-compliant schemas in order to promote and develop anesthetic specificity in data standards and augment the HL7 model for continuous quality improvement in anesthetic patient care.

From these researches, we can conclude that the common structure of the anaesthesia systems will include XML.

Most of the information in AIMS is time related. In section 2.2, we have indicated the reasons of attaching four different time dimensions to the information. The four time dimensions are also crucial for anaesthesia data because of medico-legal reasons and accuracy.

Anaesthesia data validity period consists of pre-operative, intra-operative and post-operative phases of anaesthesia. The validity starts with the pre-operative period and ends with the post-operative period. Availability time and transaction time are important because of medico-legal aspects such as identifying malpractice. Event time of a fact also helps to investigate the physician's decisions. All of the four time dimensions are necessary for storing and analyzing anaesthesia data accurately which will be needed in educational researches, anaesthesia audits and medico-legal courts.

In order to support time related data in anaesthesia information systems, user-defined times have to be used in the current non-temporal database systems. Instead of using user defined times, a temporal database system will bring advantages in terms of data accuracy and efficiency. In the next chapter, we will present our temporal XML data model and its storage structures for anaesthesia information systems.

3. PROPOSED DATA MODEL AND ITS PHYSICAL IMPLEMENTATION

3.1. Introduction

In chapter 2, we identify the need of using four different time dimensions in a temporal XML data model for Anaesthesia Information Systems.

In this chapter, we present the design of a logical temporal XML data model and its physical storage structures. Firstly, we present an anaesthesia data model in XML format. Secondly, we identify the requirements of a logical temporal XML data model for anaesthesia information systems, which supports valid time, transaction time, event time and availability time. Thirdly, we identify the needs of a numbering scheme of a temporal XML data model, and choose an appropriate numbering scheme, which eliminates the need of relabelling of nodes of temporal XML documents. Fourthly, we propose storage and index structures to store and query temporal XML data efficiently. Finally, we classify and identify the temporal query types of an anaesthesia information system and describe the way of processing temporal queries using the proposed data structures. In the next chapter, the detailed performance study results will be given.

3.2. Anaesthesia Data Model

In today's healthcare institutions anaesthesia data are stored in variety of systems. As we mentioned in section 2.6, the next generation of AIMS systems will have a common structure for their well-formed anesthesia XML documents. An XML Schema is proposed in the literature [7] for this purpose and there is research group in HL7, which is currently working for a HL7 compliant XML Schema for anaesthesia documents. When we analyze the anaesthesia documents that are currently used in the healthcare institutions, we can easily identify the parts that contain time-related data. A sample anaesthesia record is shown in Appendix D section and the figure 3.1 displays the common parts of the anaesthesia records [7].

Cast	Demographics of subjects and workers involved
Commission	Institution, client, proposed procedures, etc.
Case Pattern	Stages which make up the episode
Medical history	Medical history topics such as 'asthma'
Family history	Family History topics
Obstetric history	Index subject's obstetric history
Medication history	Regular medication taken by the index subject
Lifestyle history	Patient's lifestyle, e.g. smoking
Examination	Examination findings or measurements
Investigation	Investigation findings or measurements
Risk assessment	Risk assessments
Discussion	Risks discussed with the patient or relative
Anaesthetic plan	Explicit anaesthetic plan, if any
Translocation	Patient transit (e.g. ward to operating theatre)
Instruction	Instructions issued, e.g. pre-anaesthetic drugs
Role assignment	Roles played by workers in respect of services
Cannulation	Cannulation procedures
Monitoring	Monitoring procedures, including the gases used
Airway	Airway manoeuvres, such as intubations
Ventilation	Ventilation procedures, including the gases used
Sedation	Sedation episodes, if any
General anaesthesia	General anaesthesia episodes, if any
Regional anaesthesia	Regional anaesthesia procedures, if any
Pharmaceuticals	Drug and fluid deployments
Artefacts	Miscellaneous artefact deployments
Special procedures	Special procedures, e.g. tourniquets
Client Narrative	Surgical procedures performed
Outcome	Outcomes such as adverse results
Incident reports	Details of critical incidents, if any
Commentary	Annotations, e.g for education purpose

Figure 3.1. Topics covered by a schema for anaesthesia records.

Modern monitoring systems and anaesthesia equipments can output and then send electronic data to the anaesthesia records. Most of the data in figure D.2 are collected from the anaesthesia equipments. Data in some of the sections such as Case data, Gases and Agents, Infusions, Input-Output and other measurements are collected periodically from anaesthesia equipments and monitoring systems. Pre-operation lab results section or patient information in figure D.1 can be either taken from the information system or entered manually by the doctors. On the other hand, some parts of the records are directly taken from the anesthesia providers or surgeons. Past medical history, allergies, current medications, comments, airway management, regional anaesthetics, procedures and anaesthetic techniques are the parts that are taken from the doctors.

The anesthesia data is either implicitly or explicitly related with time. For example, case start and end times of surgery and anaesthesia, induction, intubations or the periodically measured data in the case data part, are attached with timestamps explicitly.

This does not mean that, data in the remaining parts of the anaesthesia records are not related with time. The four time dimensions that are introduced in previous chapter, have to be attached to the whole data in the anaesthesia records in order to store the validity of data, transaction interval of data, availability time of anaesthesia data and decision time of the data.

Transaction time of anaesthesia data is generated by the system. Availability time is entered by the doctors, but if the availability time is not provided, then this means that it is same with transaction time and so it is also generated by the system. Validity period of anaesthesia data is categorized into two groups. If the data is periodically measured by anaesthesia equipments then the system may generate the validity period of the data because the next measurement will be done at a predefined time in the future. However, if the data is taken from the doctors, then the validity period of the anaesthesia data is also has to be taken from the doctors. Event time of the anaesthesia data has to be explicitly defined by the doctors because it shows the relationship of the events and the system cannot be aware of these relationships. Figure 3.2 and 3.3 display a part of an anaesthesia record in XML format.

```
<patient>
  <surgery>
    <pre-operative period> ..... </pre-operative period>
    <intra-operative period>
      <case time>
        <anaesthesia start =18.33 end=19.07>
        <surgery start =20.11 end=20.20>
        <induction  start=18.47/>
        <intubation  start=18.50/>
      <case time>
      <patient information>
        <id> 1212345</id>
        <weight> 68.2 kg</weight >
        <height>173 </height>
        <asa>3 Elective </asa>
        <preopvitals> BP-160/88 HR-78 RR-24   Temp-37.0 </ preopvitals >
      <patient information>
      <regional anaesthetics>
        <block 1> Epidural</ block 1>
        <time> 18.45</ time >
        <needle> 18 GA 3.5" HUSTED  </ needle >
        <location>L3-4 </ location >
        <position> Sitting </ position >
        <amount>3 CC test dose </ amount >
        <local> .5% BUPIVACAINE + EPI  </ local >
        <comment> .5% BUPIVACAINE + EPI  </ comment >
      < /regional anaesthetics >
      <Losses>
        <bloodLoss time=19.15><amount>150</amount></ bloodLoss >
        <bloodLoss time=20.20>><amount>200</amount></ bloodLoss >
        <urine output time=19.10>><amount>150</amount></ urine output >
        <urine output time=19.15>><amount>25</amount></ urine output >
        <urine output time=20.20>><amount>100</amount></ urine output >
      < /Losses >
```

Figure 3.2. Sample Anesthesia Record- Part 1

```
        <CaseData>
          <Drugs>
            <Drug>
              <name>Ephedrine</name>
              <amount time=18.55>
                <unit>mg</ unit >
                <value>10.00</value>
              </amount>
            </ Drug >
            <Drug>
              <name>Fentanyl</name>
              <amount time=18.35>
                <unit>cc</ unit >
                <value>1.00</value>
              </amount>
              <amount time=18.45>
                <unit>cc</ unit >
                <value>9.00</value>
              </amount>
            </ Drug >
          </ Drugs >
        </CaseData>
      </intra-operative period>
    <post-operative period>.....</post-operative period>
    </surgery>
</patient>
```

Figure 3.3. Sample Anesthesia Record- Part 2

3.3. Logical Data Model for Temporal XML Documents

In the previous section, we identify the parts of the anaesthesia data model. In this section, we present a temporal XML data model, based on the XPath data model [23], that we propose for the representation and management of time-related anaesthesia information. The model, TXML, supports multiple temporal dimensions that have to be attached to the information in order to store and query data accurately.

A detailed explanation of XPath data model is given in Appendix C. A temporal XML data model has additional properties than an XML data model resulting from its temporal dimensions.

There are two steps to represent time-related data in a temporal XML data model. The first step is to group the versions of the time-related element together. A new tag <group></group> is used for the elements that may have different values as the time progresses. The second step is to store temporal information of an element. We have to represent the temporal information somewhere in the model. There are two ways to store temporal information in the model. First one is to attach time information to edges, and the second one is to define time elements as subelements. The former is implemented in some of the proposed temporal XML data models in the literature [36] [40]. However, it can only be applicable in the temporal data models that do not support transaction time. In some cases, when we want to delete a node because of a mistake in its valid time interval, the node is not deleted instead it is transaction time end point is set to the deletion time and a new time element is attached to the same node in a temporal data model which supports transaction time. This means that, in a valid time update that requires logical deletion and insertion we have two different time elements for the same node. We can categorize updates into two: Timestamp update and Value update. In timestamp updates, we attach a second time element to the current node. However in value updates, a new node created with a new value and a new timestamp element. In value updates, the new node is inserted into the group tag. The mentioned XML data models that attach time information to the edges represent the time information like an attribute of that element. However, attributes must be unique inside the elements. If the number of time information for a node is more than one, then we cannot use timestamps as attributes. A better solution, that does not modify XPath data model, is proposed in [37]. The proposed data model supports three different time dimensions, which are valid time, transaction time and efficacy time for normative text documents, and represents them as sub-elements in the main element.

In our proposed temporal XML data model, TXML, we define a time element that consists of four different time dimensions as its subelements. Each of the sub-elements contains "*low*" and "*high*" attributes to represent start and end point of their time intervals. Figure 3.4 and 3.5 shows the representations of time elements in a temporal XML document and its corresponding tree.

```
<Group>
<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 >
</ Element1>
</Group>
```

Figure 3.4. Representation of Time Elements in a temporal XML document.

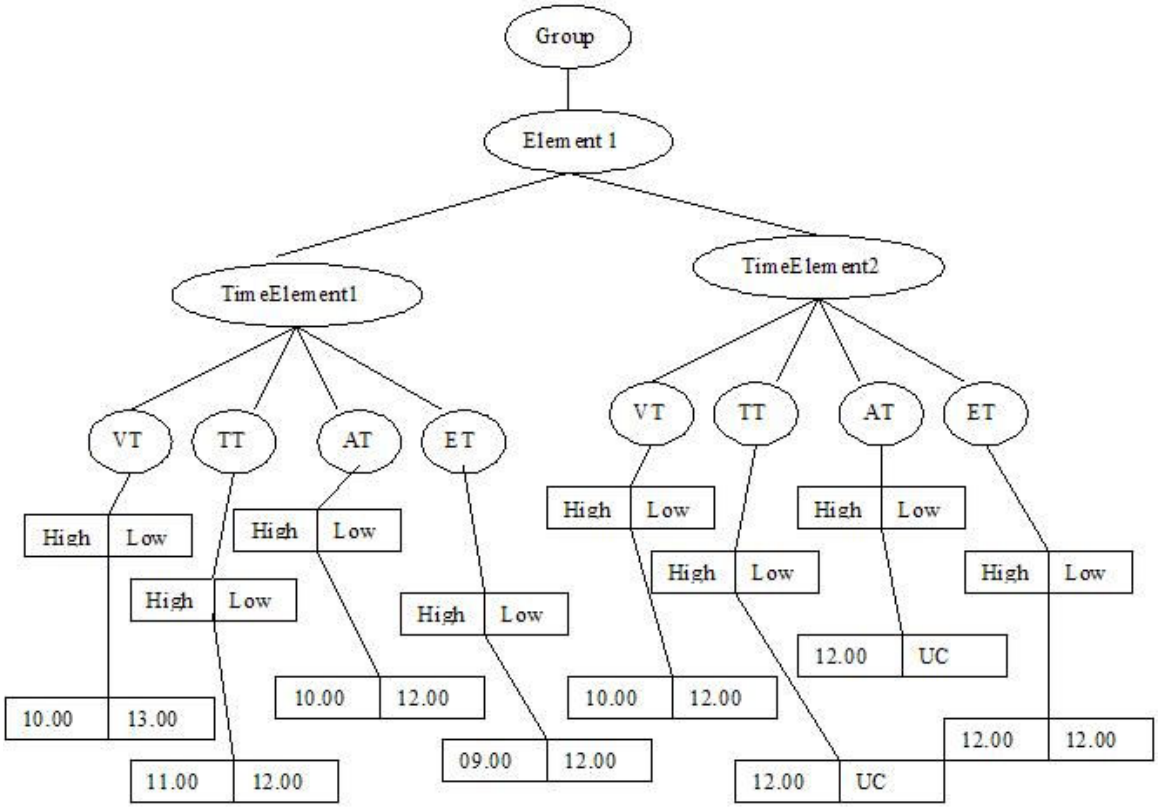


Figure 3.5. Representation of Time Elements in a temporal XML document tree.

In TXML, only nodes have time information. The edge set of a document tree is same, as in the non-temporal data model. This does not mean that edges are not associated with time information. The time of an edge is in fact determined by the time elements of the nodes at the edge's two ends. This property is also indicated in the previously defined temporal XML data models.

Adding time dimensions will bring some constraints to the data model. When a new node is inserted to the document tree, the validity interval of the parent node must start before the new node's validity interval. If the validity of the child node starts before the parent node, then the validity interval of the parent node is re-calculated and expanded to the union of validity intervals of its children and descendants. When a node validity period is terminated then all of its children and descendant nodes validity interval must also be terminated. Validity interval of the root node in the document is the union of validity periods of all nodes in the document.

When a new node is inserted to the database, transaction time interval start point is set to the insertion time. Since a node can not be inserted to a tree if its parent is not already inserted, transaction time start point (TT_S) of a new node must always be greater than its parent's transaction time start point, i.e. parent.TT_S <= child.TT_S. When a node is logically deleted from the database, then all of its children and descendants must also be logically deleted (if parent.$TT_S = t_d$ then child.$TT_S = t_d$).

If a node becomes available to the information system at AT_S, then all of its children and descendants can be available to the information system at AT_S or later. When the information system realizes the data is incorrect at time AT_E, then this means that all of its descendants are also incorrect and their availability time end point is set to AT_E.

Initiating event time of a node is equal or smaller than all initiating times of its children and descendants.

Practically, it is troublesome to include all time elements in a document tree. Timestamps of a node are inherited by its descendants, unless redefined. Redefinitions are

done if the timestamps of an element is different from its parent's timestamps. If a node does not contain timestamps then its parent's timestamps are used for the child node.

Finally, the timestamps of grouped elements must be disjoint, that is for each node at most one time element is associated to any four-dimensional time point.

Similar relationships (except the event and availability time properties), which provide temporal consistency, are also defined by other authors [36][37] [40].

After identifying the properties of our temporal XML data model, the next step is to apply it on the anaesthesia data model. The four time dimension are added as subelements under the to anaesthesia elements. The element types that can have multiple values over time are grouped under the group elements, so that if we want to extract the history of that element type we can easily access it. Figure 3.6 displays a part of the anaesthesia record document which is shown in figure 3.2 and 3.3 in a temporal XML document. Representing temporal information of the remaining elements will be done in a similar way.

```
<patient>
  <TimeElement id=1>
    <VT Low=01.12.2006 High=02.12.2006></VT>
    <TT Low=01.12.2006 High=UC></TT>
    <ET Low=30.11.2006 High=30.11.2006></ET>
    <AT Low=01.12.2006 High=UC ></AT>
  </ TimeElement >
<surgery>
  <intra-operative period>
    <TimeElement id=2>
      <VT Low=01.12.2006 High=01.12.2006></VT>
      <TT Low=01.12.2006 High=UC></TT>
      <ET Low=30.11.2006 High=30.11.2006></ET>
      <AT Low=01.12.2006 High=UC ></AT>
    </ TimeElement >
    <Losses>
    <group id=1>
      <bloodLoss>
        <TimeElement id=3>
          <VT Low=01.12.2006 19.15 High=01.12.2006 19.15 ></VT>
          <TT Low=01.12.2006 19.15 High=UC></TT>
          <ET Low=30.11.2006 19.15 High=30.11.2006 19.15 ></ET>
          <AT Low=01.12.2006 19.15 High=UC ></AT>
        </ TimeElement >
        <amount>150</amount>
      </bloodLoss>
      <bloodLoss>
        <TimeElement id=4>
          <VT Low=01.12.2006 20.20 High=01.12.2006 20.20></VT>
          <TT Low=01.12.2006 20.20High=UC></TT>
          <ET Low=30.11.2006 20.20High=30.11.2006 20.20></ET>
          <AT Low=01.12.2006 20.20 High=UC ></AT>
        </ TimeElement >
        <amount>200</amount>
      </bloodLoss>
    </Losses>
  <intra-operative period>
</surgery>
</patient>
```

Figure 3.6. Temporal XML document of a sample Anesthesia Record

3.4. Proposed Storage Structures

In the previous section, we propose a logical temporal XML model for anaesthesia data. Special data structures are needed to support temporal data storage. In this section, we propose data and index structures for implementing the temporal XML data model. As we briefly explain, in the subsection 3.4.1, we present a numbering scheme that we used to support node accesses in the structures. In subsection 3.4.2, we present a schema index table, for extracting paths of the temporal XML documents from the schema tree. In the subsection 3.4.3, we present an address table for storing the location of nodes in the data model. The address table will be used for accessing nodes of the model. In subsection 3.4.4, we propose the basic storage structures that store the elements of the temporal XML documents and the contents of the elements and their timestamp elements. In subsection 3.4.5, we introduce time index structures in order to support temporal queries efficiently.
In subsection 3.4.6 and 3.4.7, we present two index structures for supporting selection and join operations. All of the index structures in this section are proposed to eliminate the need of traversing document trees. The proposed time index structures help to process various temporal query types efficiently.

3.4.1. Numbering Scheme

Indexing XML documents will eliminate the need of accessing actual documents. To make this possible, a unique label is assigned for each node in the XML trees, which can show the relationship between any two nodes. As mentioned in [53], several node-labelling techniques have been proposed in the literature [54] [55]. All of these techniques help to improve query performance. If deletion and/or insertion occur regularly in the XML data, these techniques would need expensive re-computing of existing labels. However, frequent updates will take time and reduce the query performance. All of the index structures that consists the precomputed nodes are also updated.

The number of updates in temporal XML documents is greater than non-temporal XML documents because of time dimensions. A numbering scheme, which requires re-labelling will be inefficient for temporal XML documents. A new labeling scheme for

dynamic XML data (LSDX) is defined in [53]. LSDX supports the representation of the ancestor – descendant relationship and sibling relationship between nodes and updating XML data without the need of re-labelling existing labels. Instead of using only numbers or letters, LSDX combines both numbers and letters to label XML trees. One of the advantage of using both numbers and letters, is labels are compact and persistent. When a new node is inserted, there is no need to re-label the existing nodes in LSDX. Figure 3.7 displays a sample xml document tree labeled with LSDX numbering scheme.

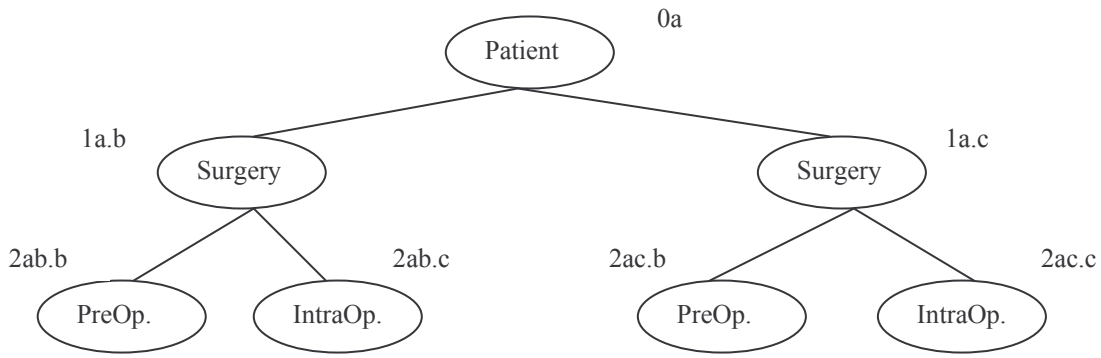


Figure 3.7. Sample XML document tree labeled with LSDX numbering scheme.

In general, LSDX labels nodes as follows: Given a node v with n child nodes: u_1, u_2, u_3, ... u_n, a unique code for u_1 is a combination of its level + code of its parent node + "." + "b". The unique code for u_2 is its level + code of its parent node + "." + "c". The labeling of nodes continues for the rest of child nodes in alphabetical order.

Some of the labeling techniques reserves extra number spaces or preserve codes. However, when all of the reserved spaces and codes are consumed, relabelling of nodes is needed [53]. LSDX combines numbers and letters and eliminates the need of relabelling.

The rule of generating new labels is described in [53] as follows: If there is no node standing before the place that a new node will be added, get the code of the node standing after the new node and insert "a" after the ".". Otherwise, keep counting from the code standing before it so that the code for the new node will be greater than the code of its

previous sibling and less than the code of its next sibling in alphabetical order. If previous code ends with “z”, attach “b” at the end.

All of the advantages of LSDX that are explained in this section make LSDX a good choice to label nodes in temporal XML documents.

3.4.2. XML Schema Tree Index

In order to extract possible paths of the element types in the XML schema documents, an index table is constructed from the schema tree. A similar idea for extracting paths is proposed in [51]. Figure 3.8 displays a sample XML schema tree of an XML document, which stores anaesthesia data.

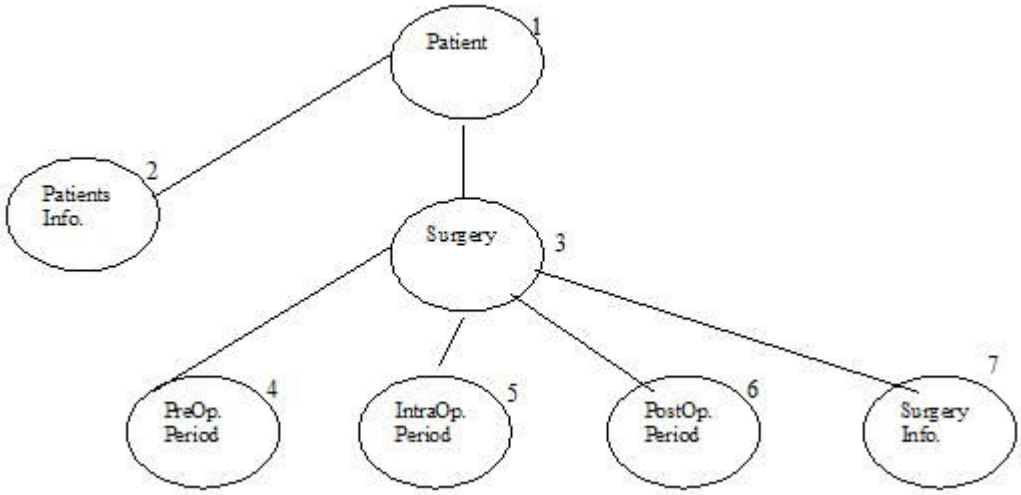


Figure 3.8. Sample XML schema of Anaesthesia document

A schema index table is constructed to extract paths from the XML schema. Each element or attribute in the XML Schema is represented by a row in the table that consists Node ID, Node Type, Parent ID and Parent Element Type fields. Element type field is not unique since there can be same element types that have different paths reaching from root of the tree.

Table 3.1 shows the schema index of the sample schema tree represented in Figure 3.8.

Table 3.1 Schema index table

Element Type	Element ID	Parent ID	Parent Element Type
Patient	1	Null	Null
Patient Info	2	1	Patient
Surgery	3	1	Patient
PreOp Period	4	3	Surgery
IntraOp Period	5	3	Surgery
PostOp Period	6	3	Surgery
Surgery Info	7	3	Surgery

The following algorithm describes how to extract the paths of an element in the XML schema tree.

Extract Path Algorithm:

```
Input: element type
Output: result: possible paths from root to the input element type.
Path info is in the documentId/rootId…./InputelementId format.
Begin
        For each row whose element type is the input element in the schema index table Do
                Element Id ← parent id of the row
                Element type ← parent element type id of the row
                If (Element Id == Root)
                        Return Element Id;
                Initialize PathInfo;
                While (Element Id  != Root)){
                        PathInfo=Element Id + "/"
                        New Row=ExtractRow (Element Id, Element Type);
                        Element Id ← parent id of the new row
                        Element type ← parent element type id of the new row;
                }
                Return documented+"/"+PathInfo;
        End
End.
```

For example, we want to find the path of "PreOp Period" element. For each of the "PreOp Period" element we repeat the algorithm. In this case, there is only one in the

schema index table, which is the fourth row. After executing the "for" part of the algorithm the next step is extracting the parent element information in that row. The parent element type is "Surgery" and its id is 3. Then using this information we search the table for the row in the table whose element type is "Surgery" and element id is "3" and we reach the third row. From the third, row we extract the parent of the Surgery element, whose element type is "Patient" and element id is 1. Using this information we reach the first row of the table, and extract the parent of the "Patient" element. However, the parent of the "Patient" element is null and this means that we reach the root so that algorithm stops here. The path information of the PreOp Period element is AnaesthesiaDb/Patient/Surgery/PreOp Period. Path information of the element types in XML Schema documents, which is defined in the definition 1, will be used as *pathInfo* throughout this thesis.

Definition 1: Path of an element type in an xml schema document gives the path from the root to the element type. PathInfo of an element e is defined as documentId/rootId/e_1*Id/.../*e_n*Id/elementId where* e_i *is the ancestor element of e and (depth of e -1) => i >= 1.*

3.4.3. Address Tables

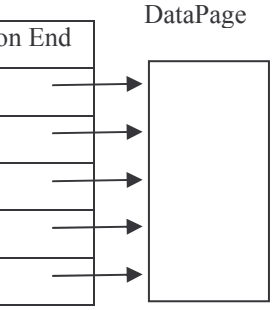
There are two ways to handle relationships between nodes. One is to use node numbers of the referenced node and the second one is to use a pointer. By using pointers, we can directly reach the referenced nodes. For example, the parent of a node can be reached by a parent pointer stored in the node structure. However, if the parent location is changed or the parent node is deleted, then it will lead dangling pointers problem [56]. Using node identifiers instead of pointers eliminates the dangling pointers problem. When a node memory location is changed then we need to update all the locations that use a pointer to that data. If the node identifiers and their memory locations are stored in a single structure and update are done in this structure then it will be much more reliable than pointers. The address table is sorted according to node identifiers. We can define an address table as follows:

Definition 2: An address table is a structure that stores unique node identifiers and their memory locations. The information consists of the document identifier, node identifier, memory location and a pointer to the data page.

Table 3.2 shows a sample address table.

Table 3.2. Sample address table

Document Id	Node Id	Memory Location Start	Memory Location End
D1	1	#1	#19
D1	2	#20	#39
D1	3	#40	#59
D2	4	#60	#79
D2	5	#80	#99



3.4.4. Node Types

The proposed temporal XML data model is consists of three different kinds of nodes. In native XML database models, which are mentioned in the subsection 2.5, the main components are generally path indexes and storage structures for node types. Node types are basically categorized as internal nodes (elements) and external nodes (contents). Two of the node types in the proposed temporal XML data model are also internal nodes and external nodes. The structure of internal and external nodes is similar to the ENAXS native xml database system [51]. Third node type is introduced as *Timestamp nodes*. Since the time dimensions are represented with a timestamp element in the temporal XML document, a new kind of node type, which efficiently stores the time dimensions, is proposed. In the proposed temporal XML data model, the number of time elements that a node may contain is more than once. Each time element has four different sub-elements and their high and low attributes. To store time elements of a node in index the structures may result in multiple updates of the same time element. Instead of storing time elements in the index structures, a better solution is to store time elements of a node in the same page, since temporal queries may result frequent accesses to the time elements.

3.4.4.1. Internal Nodes. All of the nodes except the timestamp and the content nodes in the XML document tree are stored in the internal node format. The number of internal children or timestamp nodes of an internal node can be more than one, where the number of external child node can only be one. If the internal node does not contain any timestamp node, then it means that the time dimension values of that node are same as its parent. This is a kind of compression and the timestamp information that can be extracted from parent node and are not stored in the database separately. Figure 3.9 shows the internal node structure. A unique document identifier is assigned to each document and a node identifier to each node except external nodes so that a node can be identified by a pair (document identifier, node identifier) within the storage. PathInfo field is the path of the node starting from the root node. The path info 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. Node identifier is unique to each node and is assigned to a newly inserted node according to the numbering scheme defined in subsection 3.3.

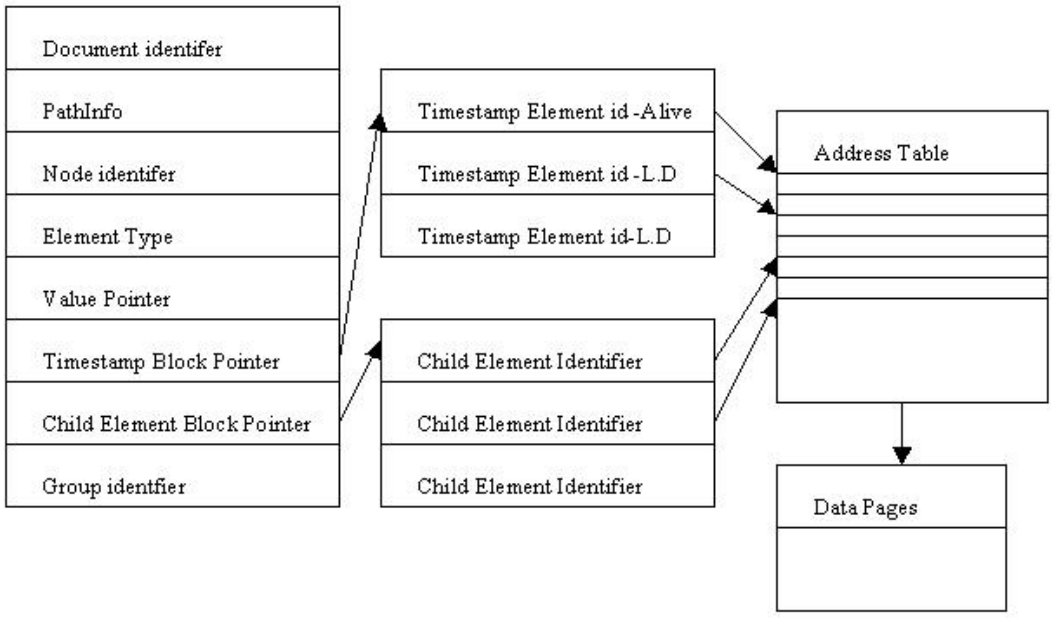


Figure 3.9. Structure of internal nodes

Element type field indicates the element type of the node. Value pointer is pointing to the external node of the node if the node has content; otherwise the value pointer is Null. Timestamp block pointer field is pointer to a timestamp block since the number of timestamp elements of a node can be more than one. If one of the timestamp fields of a node is updated, the record is deleted and inserted to the database again. Although the node is same, there are two timestamp elements of it, one is logically deleted and the other one is alive. In order to follow the history of an element, the timestamp elements must be the children of the same node. Since the number of timestamp elements is more than one, a single pointer cannot be used to represent timestamps node. Timestamp block pointer of a node, points to the alive element in a block that consists of all timestamp elements of that node. Child block pointer field is similar to the timestamp block pointer since both fields point to a block of elements. The types of the elements in child block are same as the node in this case, i.e. they are also internal nodes. The child element block holds the node's children. Both of the blocks contain node identifiers and a pointer to address tables. The reason of pointers pointing to the address tables is the dangling pointer problem [56]. Group identifier field is designed for the grouped nodes. When an element's value changes over time then the element is enclosed with a group tag. The group identifier field helps to access the nodes that are in the same group, i.e. the history of the element.

There are important two points for the storage strategy of the internal nodes. The first one is to store the internal node and its timestamp elements together. The reason behind this is while executing temporal queries we need to access the timestamp nodes of the elements frequently. The second point in the storage is to store nodes and their children in closer locations if it is possible.

3.3.4.2. External Nodes. External nodes are designed to store the content of the internal nodes; they are actually equal to the value nodes in the XML document tree. The external nodes are simpler than internal nodes since they do not have any timestamp nodes or child element node. Figure 3.10 shows the structure of the external nodes. Document identifier field identifies the document of the value node. 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. Parent pointer points to the parent of the value node in the address table. Value field contains the content of the value node.

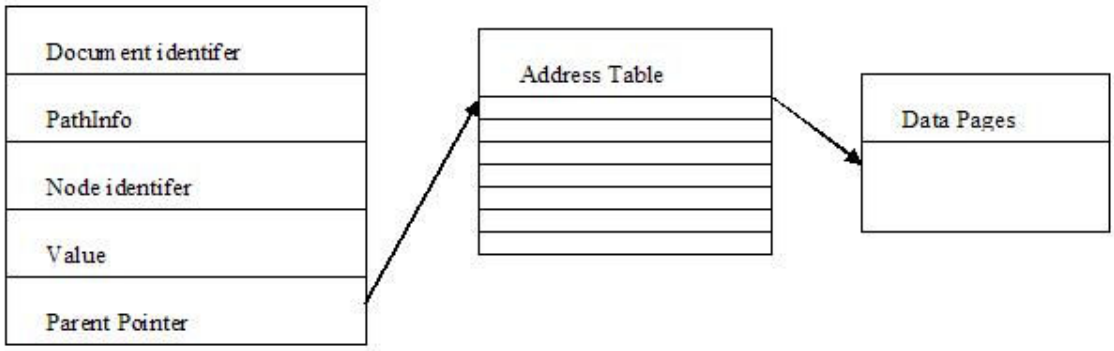


Figure 3.10. Structure of external nodes

3.3.4.3. Timestamp Nodes. Timestamp node types are designed to store timestamp elements efficiently. Each internal node may have one or more timestamp elements. We have four different temporal dimensions in the temporal XML data model. These time dimensions are stored in the timestamp element as sub-elements. VT sub-element stands for valid time dimension, TT stands for transaction time dimension, AT stands for availability time dimension and ET stands for event time dimension. In order to make the subelements HL7 compatible, *high* and *low* attributes are defined in each of them. 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. Figure 3.11 shows the structure of the timestamp nodes. Document identifier field identifies the document of the timestamp node. 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. Status field represents the state of the timestamp. The timestamp can be alive or logically deleted. Pointer to the next time stamp field points to the next timestamp of its parent node since the internal nodes may have more than one timestamp node.

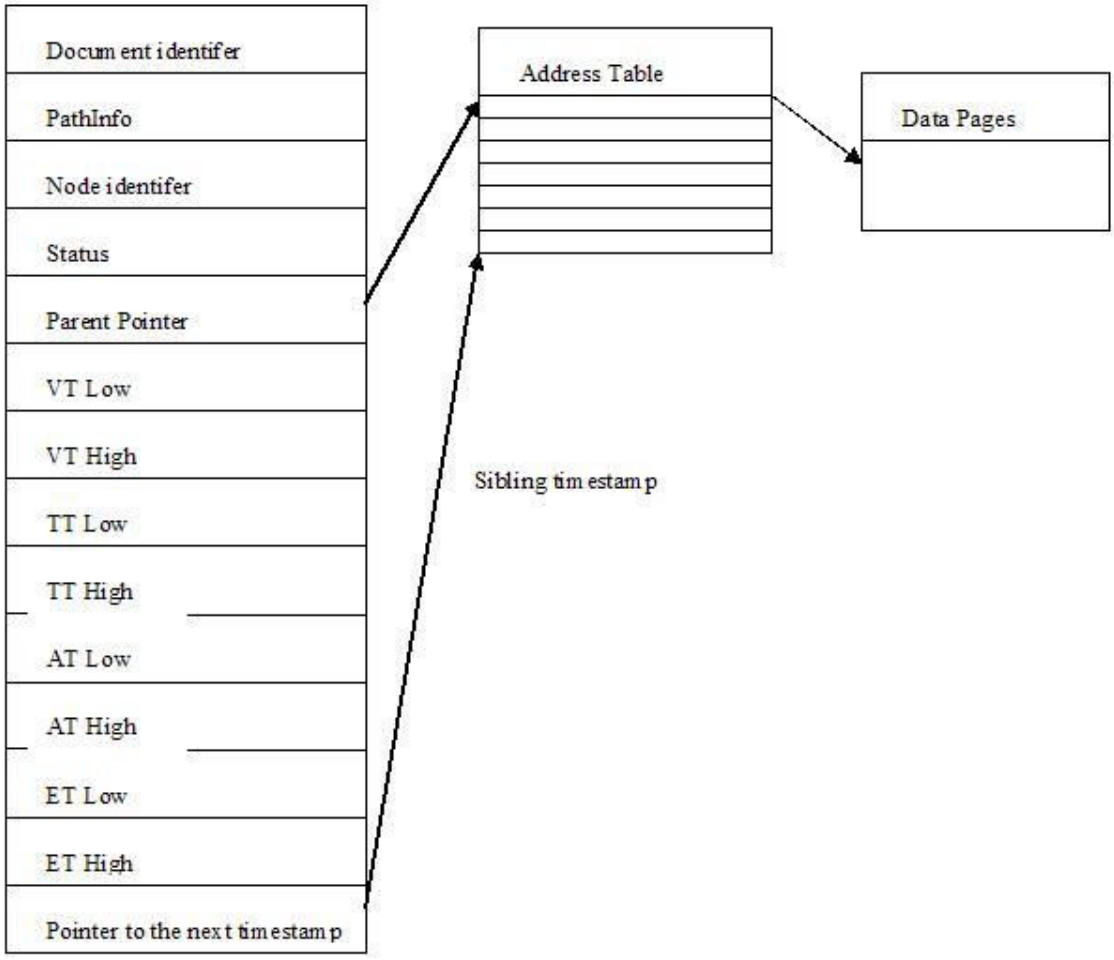


Figure 3.11. Structure of timestamp nodes

3.4.5. Time Index

Time related data have to be stored in special data structures so that it can be efficiently and accurately stored and queried. In the previous section, some of the multidimensional indices that are defined in the literature are introduced. However, neither of them is designed to support four time dimensions. Time dimensions have different properties than spatial dimensions because of their now-relative property. In order to use multidimensional indices for multiple time dimensions, modification is needed. As mentioned in [49], most of the existing multidimensional indices are designed for all dimensional queries. However, temporal queries do not always use all of the time dimensions; instead they use different combinations of the four time dimensions. In the

literature, AR* tree [49] 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 pre-operative and post-operative anaesthesia periods. Generally, when we insert a data to the database and some time later realized that it is incorrect, we delete the data. However, anaesthesia data must not be deleted because of medico-legal reasons. If anaesthesia data is current in the database then the end point of its transaction time dimension is equal to now. Availability time has similar characteristics with transaction time, i.e. if the data is current then availability end point is also equal to now.

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

3.4.5.1. Temporal AR* Tree. Temporal AR* tree is described in this subsection including its structure and insertion, deletion and search algorithms.

Temporal AR* tree is an index structure which combines the partially dimensional property of AR* tree and temporal property of Double-R tree (2R-tree). 2R tree is chosen as the multidimensional indexing method because it supports now relative data for transaction time dimension. Anaesthesia data is not now-related in terms of valid time dimension. There is a proposed index structure that supports now-relative data for both valid and transaction time [45]. However, one of the disadvantages of this method is, it does not use off-the shelf methods and it is very complex. It adds additional heuristics based on bitemporal regions to the original R* tree. Since anaesthesia data is valid between pre-operative and post-operative periods, it is valid time end interval is closed. Because of this reason a multidimensional temporal index structure, which supports now-relative transaction time, is suitable for anaesthesia data. The temporal data model that supports

four time dimensions, where transaction time and availability time is now relative, is needed.

In the literature, there are some proposal that support now-relative transaction time such as Bitemporal R tree and Double R tree [50]. Bitemporal R tree is a multi-root R tree, which stores the states assumed by the ephemeral R-tree through its transaction time. Bitemporal R tree is composed of several logical R trees, representing the evolution of objects in the transaction time. Although Bitemporal R tree is efficient for transaction time timeslice queries, because of its structure, it is not possible to add a now-relative time dimension, i.e. availability time, to Bitemporal R tree. As mentioned in [50], a second method which is based on double R tree method, 2R tree is a good alternative to Bitemporal R tree and can be extended to support four time dimensions because of its structure.

As in the case of 2R tree, there are two R trees in the structure. The front R tree holds the alive objects and the back R tree is designed for storing logically deleted objects. Each of the front and R tree supports partially dimensional queries by using node groups. Data and its enclosing MBR are represented by [VT_S, VTE, ET_S, ET_E, TTS, TT_E, AT_S, AT_E]. When a data is inserted to the database, since it's alive and transaction and availability time end points are UC, it is inserted to the front R tree. If the same data record is deleted, i.e. logically deleted, from database then the record is deleted from front R tree and is inserted to back R tree while setting its transaction end point to the deletion time and availability time to a specific time which is taken from the user. When we analyze the transaction time and availability time relationship, we see that data can be inserted only when it is known and data can be deleted only when it's recognized as incorrect, i.e. $AT_S <= TT_S$ and $AT_E <= TT_E$. For this reason, when AT_S is a time point then TT_S is also a time point although they are not necessarily equal and if AT_S is equal to UC then TT_S is also UC. We can conclude that both availability time and transaction time sends a data object to the same tree.

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

time intervals. Data 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 TAR* tree. Original 2R tree uses only valid and transaction times where data is indexed by its transaction start time point in the front R tree. In TAR* tree valid time and transaction time dimensions are extended with event and availability time dimensions. The structure of the TAR* tree is depicted in figure 3.12.

When an object is inserted to the database, it is inserted to the front R tree. As mentioned in the AR* tree [49], insertion algorithm is the naive extension of the original R* tree. 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. The important point in here is the front R tree indexes the data objects using only start points of their availability and transaction time dimensions. In order to delete a data object, i.e. logical deletion, two 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 alive records can be deleted and all alive records are stored in the front R tree. Search algorithm is invoked to find the data object in the R tree. Details of the search algorithm will be given later in this subsection. Then the data object is deleted from the front R tree and is inserted to the back R tree with a little difference. 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 taken from the user. If the user does not supply availability time then it 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 equals to deletion from the front R tree and insertion to the back R tree. The deletion algorithm that is used in the front R tree is a naive extension of the original R* tree as mentioned in the AR* tree [49].

The combination of four different time dimensions will produce in 2^4-1 query types. Node group structure of AR* tree provides efficient processing for each query type. When a query that does not have any constraint in transaction time and the end point of the availability time is asked, it has to be executed on the alive objects in the front R tree since the query execution time is *now*. All of the alive objects are stored in the front R tree so only the front R tree is used during a search with a query that does not have any constraint in transaction time dimension. The search algorithm of TAR* tree is similar to the AR* tree search algorithm. However, a query that has a time constraint on the end point of the

availability time or transaction time dimension means that the query is asking for logically deleted objects. So the query is executed on the back R tree which stores only logically deleted records. All of the remaining query types will be executed in the same way and have to be searched in both front and back R tree.

Execution of queries in front and back r tree is somewhat different. In back R tree, queries are executed in a usual way, where in the front R tree transaction time and availability time constraints have to be transformed before evaluating. For example if a query asks data objects that are alive at t_i and valid at v_j, both front and back R tree are searched. The back R-tree is searched for all rectangles that contain point (t_i, v_j). The front R-tree is searched for all vertical intervals which intersect a horizontal interval H. Interval H starts from the beginning of transaction time and extends until point t_i at height v_j (Fig. 2.12). The advantage of the search algorithm in both R trees is, it does not access the irrelevant dimensions while searching as in AR* tree. The following algorithm describes the evaluation of search queries in TAR* tree. The search algorithm is same as AR* tree search algorithm and is described in subsection 2.6.4. Transform query algorithm is used for front R tree. Only start point of transaction and availability time dimensions are indexed. So when a query asks for a time point t_i in transaction/availability time then the search has to be start from the beginning of transaction time and extends until point t_i. Transform Query algorithm assigns the beginning of transaction time to T_s and t_i to $T_{E.}$ General Search algorithm is the same as AR* tree search algorithm.

Search Algorithm:

Input: rectangle: query range
node-group: initial node-group of the query
Output: result: all the tuples in rectangle
Begin
If (initial node group contains T_E and/or A_E)
    generalSearch (rectangle, node-group, back R tree);
Else If (initial node group does not contain T or A)
    generalSearch (rectangle, node-group, front R tree);
Else If (initial node group contains T_S and/or A_S)
    generalSearch (rectangle, node-group, back R tree);

rectangle ‹--TransformQuery(T_S and/or A_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 (T_S)

T_E ‹-- t_i

T_S ‹-- beginning of transaction time in the tree

If (A_S)

A_E ‹-- t_j

A_S ‹-- beginning of transaction time in the tree

End

TAR* tree is designed for efficiently querying partially dimensional temporal range queries in temporal XML documents. There are two different usage of TAR* tree in the data model. In the first one we use the TAR* tree to index root to leaf paths. The number of TAR* tree in the model is equal to the number of root to leaf paths. In the second one, all non-leaf nodes are indexed by a single TAR* tree.

We have stated that the leaf nodes of the TAR* tree are composed of a pointer to the data and its enclosing MBR (MBR will simply be used instead of 4-D hyper-rectangle throughout the thesis). When a new element is inserted one of the TAR* tree, then its enclosing MBR is also inserted to path index tables and join table. The MBR relation table will be update with the new MBR. The insertion of new MBR in the index tables has been stated in the relevant subsections.

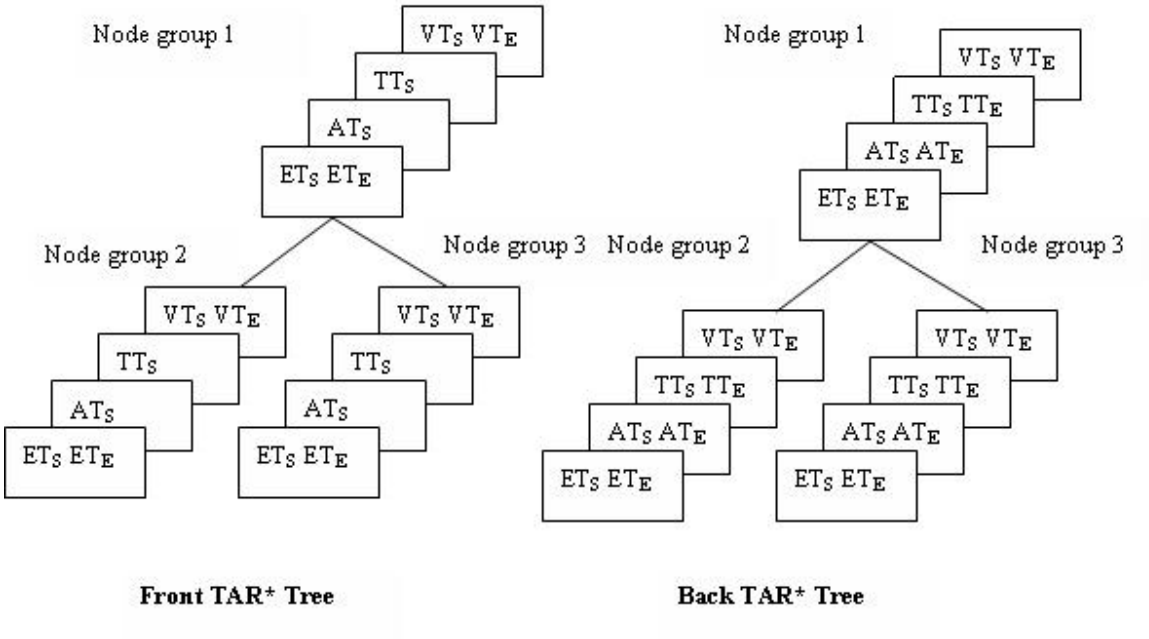


Figure 3.12. Structure of TAR* tree

3.4.5.2. MBR Relation Table. Although indexing paths of temporal documents and storing join tables helps reduce the search space; as stated in [52], computing paths within a given time interval is quite expensive even in the presence of traditional path indexes. TAR* tree is used for indexing temporal dimensions of the paths in a temporal XML document. When a query with a constraint on temporal dimensions is asked, the TAR* tree will efficiently return the result. However, TAR* tree only indexes temporal dimensions. When a query with a constraint on any of the key dimensions is asked (for example: return all D nodes where D.Aid=5 and D.Timestamp.ValidTime=2), then if we use TAR* tree to execute the query, unnecessary nodes will be accessed. As a solution, we can add the node identifiers as a new dimension to TAR* tree. However a path may contain multiple intermediate nodes and since we don't know which node type constraint will be given in the query, we have to add all 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 two nodes have the same timestamp values then they will be indexed in the same MBR. However in the second case, the two

nodes will be indexed in different boxes because of their key dimensions and will lead an inefficiency of the searches in TAR* tree.

A possible solution for this type of queries is, not to use the TAR* tree directly. Instead of searching TAR* tree using only time dimensions, we can search the path index and join index tables and apply the time constraints in that tables. MBR fields in the path index tables and join tables give the information of temporal dimensions of the nodes. But this does not prevent us to compare the list of MBR's because the records in the index tables stores only the minimum bounding boxes of themselves or their descendant. There is no information of the relationships between minimum bounding boxes. A good solution is to precompute the overlapping MBR's when a new node is inserted to the TAR* trees. This will prevent us to compare the time dimensions during query processing.

A MBR relation table is proposed to store the overlapping MBR's from all TAR* trees. Table 3.3 shows the structure of the MBR relation table.

Table 3.3. MBR Relation Table

TAR* Tree Path Type	MBRId	Overlapped MBR's
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 have stated before, the node time interval is the union of its children's time intervals. So when a leaf node is inserted to the corresponding TAR* tree, it is obvious that its MBR will be overlapped with the MBR of its ancestors in the TAR* tree of non-leaf nodes. Although MBR relation table has the advantage of precomputed overlapping intervals and will increase the search performance, it has a disadvantage of update cost.

3.4.5.3. B+ Tree Index for Difference Queries. Traditional time indices index time dimensions according to start and end point of their intervals. However, temporal queries sometimes have comparison constraints on the start and end point of the same time

dimension or start/end point of different time dimensions. For example, a query may ask complications of anaesthesia drug that appeared no later than 30 minutes after their initiating event. A second query may ask all test results, which are inserted to the database retroactively (TT_S>VT_S). Traditional time indices cannot efficiently process these types of queries. Our temporal XML data model supports four time dimensions: Valid Time, Transaction Time, Event Time and Availability Time. Although possible number of time dimension tuples is twenty-four, most of the comparison queries use start point of the time dimensions. Table 3.4 shows a sample set of the possible combinations of time dimensions. Traditional indices index the start and end point of the same time dimensions. If we index the start and end point of different time dimensions, the difference queries will be processed efficiently.

B+ trees can be used to index these time dimension tuples. The leaf nodes of B+ trees will contain the element type of the node and a pointer to the node. The query that asks the complications of anaesthesia drug that appeared no later than 30 minutes after their initiating event is executed on the second B+ tree for the complication element type. However, the open-end problem prevents us using B+ trees in the tuples that contains now-relative data. A possible solution is to use TAR* tree for the now-relative data in comparison queries.

Table 3.4. Sample Time Tuples

Type	Time Dimension Tuples
1	V_ST_S
2	V_SE_S
3	V_SA_S
4	T_SE_S
5	T_SA_S
6	E_SA_S

3.4.6. Path Index Table

As mentioned in [52], indexing paths that are valid during a certain interval rather than nodes enhances the query performance dramatically. This ability is not provided by traditional path indexes. The basic idea of our path index tables is similar to the one in [52], in order to take the advantage of indexing continuous paths rather than nodes. In [52], the authors index all continuous paths' equivalence classes 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 paths in a document will require huge number of index tables. Instead of indexing all paths, we index all root to leaf paths and create another solution for non-leaf paths, which will be described in the next subsection.

In our proposal, we define one path index table for each root to leaf path type. Figure 3.13 displays the structure of the path index tables. When a leaf element is inserted to the database then the path index table for that path 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. The NodeId field represents the unique identifier of the inserted element. 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. Status field indicates the status of the node, i.e. the node is current or logically deleted. Timestamps field consists of the timestamp elements of the node. Timestamp elements can be more than one so the field stores the node identifiers of the timestamp elements. When a new leaf element is inserted, it is also inserted to the corresponding time index. All data objects that are inserted to the time index have a MBR enclosing it. The MBR id of the element is inserted to the path table. The MBR field indicates the element's minimum bounding rectangle and will be used in temporal query processing.

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

Status, MBR List and the B + tree index are the differences of the proposed model.

Figure 3.13. Structure of the path index table

3.4.7. Join Index Table

Path index tables are used to index only root to leaf paths. Non-leaf paths are not indexed with path index tables. However, if we index all root to non-leaf paths separately, the number of index tables increases dramatically. When we analyze the queries, we realized that the non-leaf nodes are generally used for joining leaf nodes. In this subsection, we propose a join table, which indexes all non-leaf nodes in one table. Figure 3.14 shows the structure of the join table.

When a node is inserted to 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.

Element Type	Path Info	Node Id	Status	Timestamps	PathIndex Instance	MBR List

Figure 3.14. Structure of the join index table

Element type field is used for storing node's element type because different element types are indexed in the same table. NodeId is the unique identifier of the node as usual. 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. Status field indicates the status of the node, i.e. the node is current or logically deleted. Timestamps field consists of the timestamp elements of the node. Timestamp elements can be more than one so the field stores the node identifiers of the timestamp elements. A non-leaf node may have one or more leaf children nodes or descendant nodes. And a join operation on a leaf node uses an ancestor non-leaf 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. Path index instances field contains the descendant leaf element records of the join element in the path index tables. If the element type used in the path of an index table then it means that the non-leaf entry descendants are stored in that path index table. A non-leaf node in the join table stores all the children/descendants 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 root to leaf path types and the values are the (PathInfo, NodeId) tuple of the nodes in the path tables which are the child/descendant of the node in the join table. 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 table, is inserted, its corresponding path index table field is updated. Each insert to the path table requires an update in the join table for the parent or ancestor node. A node's timestamp must be greater than the union of its descendant/children timestamps, so a MBR of a child node must overlap with its parent/ancestor's MBR. MBR List field of a node in the join table stores all MBR's of its descendant leaf nodes.

We have a general TAR* tree index that stores the non-leaf nodes. When a non-leaf node is inserted to the temporal XML document tree, it is also inserted to the general non-leaf TAR* tree. The MBR id of the non-leaf node is also stored in the MBR List field. Figure 3.15 and Tables 3.5-3.8 illustrate an example on the document tree, path index table and join table relationship. Numbers are used as the node identifiers for the simplicity.

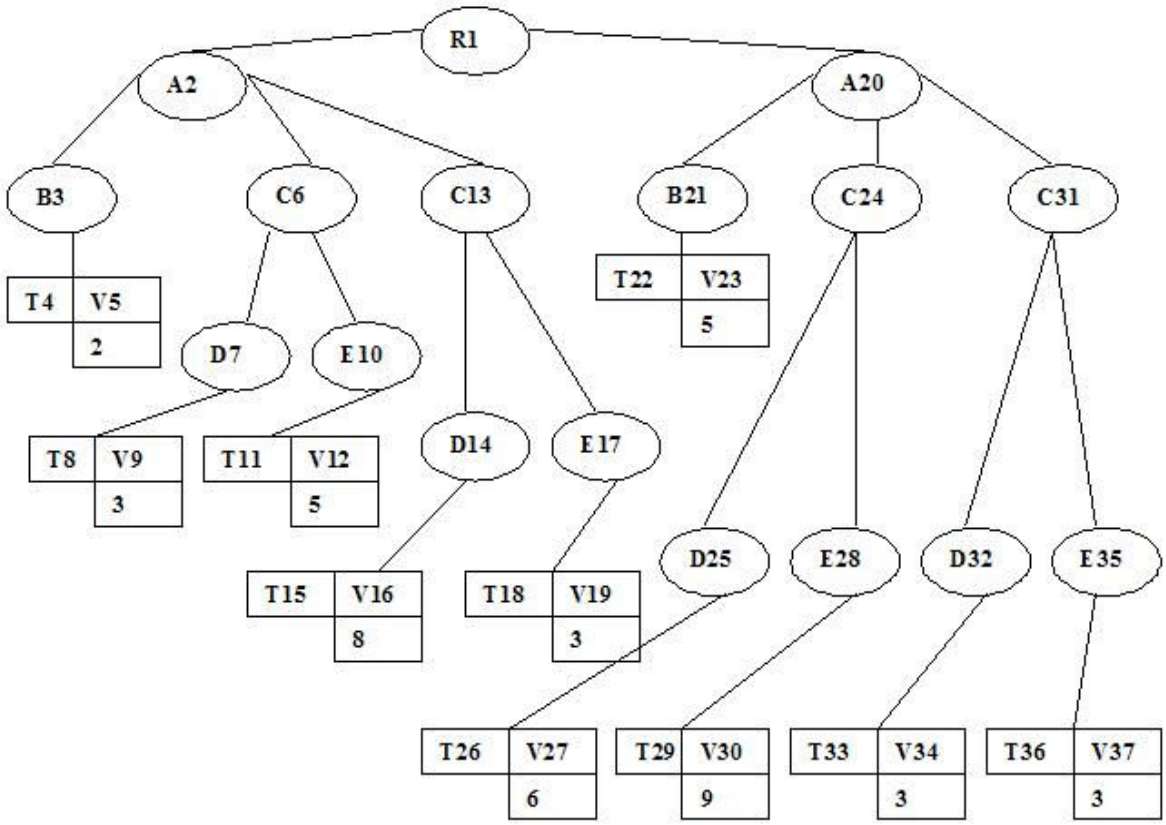


Figure 3.15. Sample temporal XML document tree

Figure 3.15 displays a sample temporal XML document. T nodes represent the time elements and V nodes represent values. When the node A2 is inserted as a child of root, since it is a non-leaf node, there will be two insertions in the index tables. The first step is the insertion of A2 to the TAR* tree of non-leaf nodes in order to index its timestamps. The returned MBR id is MBR11, which is the enclosing box of the node A2. In the next step, we insert A2 to the join table. For this moment, there aren't any children of A2 so the path child instances field will be empty. The MBR id of A2 will be inserted to the non-leaf TAR* tree key. Suppose that a new node, B3, is inserted to the document tree. B3 is a leaf node so it will be inserted to the path index tables instead of join table. In the first step we insert the B3 into the "R/A/B" TAR* tree. The MBR id if B3 is MBR1. Then we insert the node B3 into the "R/A/B" path index table. The MBR id field of B3 is set to MBR1 in the path index table. However we need to update the parent record of B3 in the join table. We know that the parent id of B3 is A2, so we update the join table and insert B3 to the child

instance field with the “R/A/B” key. The remaining nodes are inserted to the document tree in a similar way.

Table 3.5. DB1/R/A/B Path Index Table

NodeId	PathInfo	Status	Timestamps	MBRId	Value	GroupId
B3	R1/A2	Alive	T4	MBR1	5	1
B21	R1/A20	Alive	T22	MBR2	3	2

Table 3.6. DB1/R/A/C/D Path Index Table

NodeId	PathInfo	Status	Timestamps	MBRId	Value	GroupId
D7	R1/A2/C6	Alive	T8	MBR3	3	3
D14	R1/A2/C13	Alive	T15	MBR4	8	4
D25	R1/A20/C24	Alive	T26	MBR5	6	5
D32	R1/A20/C31	Alive	T33	MBR6	3	6

Table 3.7. DB1/R/A/C/E Path Index Table

NodeId	PathInfo	Status	Timestamps	MBRId	Value	GroupId
E10	R1/A2/C6	Alive	T11	MBR7	5	7
E17	R1/A2/C13	Alive	T18	MBR8	3	8
E28	R1/A20/C24	Alive	T29	MBR9	9	9
E35	R1/A20/C31	Alive	T36	MBR10	3	10

Table 3.8. Join Index Table of DB1

Element Type	Node Id.	Status	Time Stamp	PathIndex Instances	MBR List	Group Id
A	A2	Alive		R/A/B=> {(B3,R1/A2)} R/A/C/D=> {(D7, R1/A2/C6), (D14, R1/A2/C13)} R/A/C/E=> {(E10, R1/A2/C6), (E17, R1/A2/C13)}	R/A/B=> { MBR1} R/A/C/D=> { MBR3, MBR4} R/A/C/E=> { MBR7, MBR8} Non-Leaf-TAR*=> {MBR11,MBR13,MBR14}	
A	A20	Alive		R/A/B=> {(B21,R1/A20)} R/A/C/D=> {(D25, R1/A20/C24), (D32, R1/A20/C31)} R/A/C/E=> {(E28, R1/A20/C24), (E35, R1/A20/C31)}	R/A/B=> { MBR2} R/A/C/D=> { MBR5, MBR6} R/A/C/E=> { MBR9, MBR10} Non-Leaf-TAR*=> {MBR12, MBR15, MBR16}	
C	C6	Alive		R/A/C/D=> {(D7, R1/A2/C6)} R/A/C/E=> {(E10, R1/A2/C6)}	R/A/C/D=>{MBR3} R/A/C/E=>{MBR7} Non-Leaf-TAR*=> { MBR13}	
C	C13	Alive		R/A/C/D=> {(D14, R1/A2/C13)} R/A/C/E=> {(E17, R1/A2/C13)}	R/A/C/D=>{MBR4} R/A/C/E=>{MBR8} Non-Leaf-TAR*=> { MBR14}	
C	C24	Alive		R/A/C/D=> {(D25, R1/A20/C24)} R/A/C/E=> {(E28, R1/A20/C24)}	R/A/C/D=>{MBR5} R/A/C/E=>{MBR9} Non-Leaf-TAR*=> {MBR15}	
C	C31	Alive		R/A/C/D=> {(D32, R1/A20/C31)} R/A/C/E=> {(E35, R1/A20/C31)}	R/A/C/D=>{MBR6} R/A/C/E=>{MBR10} Non-Leaf-TAR*=> {MBR16}	

3.5. Temporal Query Processing Using Proposed Structures

In the previous sections of chapter 3, we described the proposed storage and index structures that are specially designed for time-related XML data. The physical structures are not only efficient for temporal queries but also for non-temporal queries. In this section, we will explain how to process both temporal and non-temporal XPath queries using the proposed data structures. We do not need to extend XPath query language, because the time element, which is introduced in our temporal XML data model, obeys the rules of XPath data model.

Supporting four different time dimensions will produce different types of temporal queries. Some of the frequently used temporal query types are: Temporal projection, temporal selection, temporal slicing, temporal join, period containment and temporal comparison queries [35] [41]. Detailed explanations of these query types are mentioned in section 2.3.

Selection

Query 1: List the patients of the surgeon "J. Smith".

XPath expression of the query is:
//patient/surgery/intra-operative/surgeons [primary="J.Smith"]
The execution of the query starts with the B+ tree that points to the "patient/surgery/surgeons/primary" path index table. Once we find the nodes that contain "J. Smith", using the pathinfo field we can extract the patient identifier and access them via address tables. B+ trees attached to the path index tables, provides direct accesses to the selected nodes.

Join
Query 2: List the urine output values of the patients who were administrated the drug "norcuron".

XPath expression of the query is:

//patient/surgery/intra-operative[//drug/name=”Norcuron”][//urine-output]

Using path table index and join table index we process join queries efficiently. From the B+ trees of the path table index, we access the matched drug elements. Pathinfo field of the elements give the id of the join element, i.e. intra-operative nodes. We access the intra-operative elements of the matched drugs in the join table. Urine output element instances are stored in the path index instance field of the join table. We access the resultant urine-output elements via address tables. Figure 3.16 illustrates the Query 2 on the node tree.

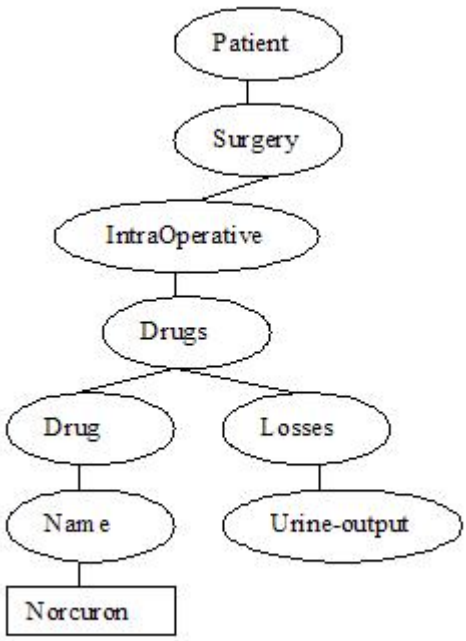


Figure 3.16. Query 2

Temporal Projection

Query 3: What is the valid time history of blood loss of the patients that are administrated the “Bipuvac” during surgery?

XPath expression of the query is:

//patient/surgery/intra-operative[/Drug/Name=”Bipuvac”]//BloodLoss/TimeElement/VT

The execution of the query starts on the "/Patient/Surgery/Intra-operative/Drug/Name" path index table. Using the B+ trees that are constructed on the values, we reach the drug elements whose name is Bipuvac. Since we access the drug nodes via path tables, we eliminate the irrelevant drug elements that exist in the other paths of the tree. From the PathInfo field of those drugs, we reach the intra-operative element ids. In the second step, we find the intra-operative elements in the join table using the intra-operative element id list in the previous step and return the descendant bloodloss elements of the intra-operative element. The last step is to extract the bloodloss histories from the "Patient/Surgery/ Intra-operative/Loss/BloodLoss" path index table. Figure 3.17 illustrates the Query 3 on the node tree. By using the B+ trees constructed on the name value of drug, we extract the drugs that match the query condition. Path index table gives the distinct intra-operative elements of the matched drugs. From the join table we extract the join elements and their bloodloss descendants. The bloodloss path index table is the last structure which gives the valid time histories of the matched elements.

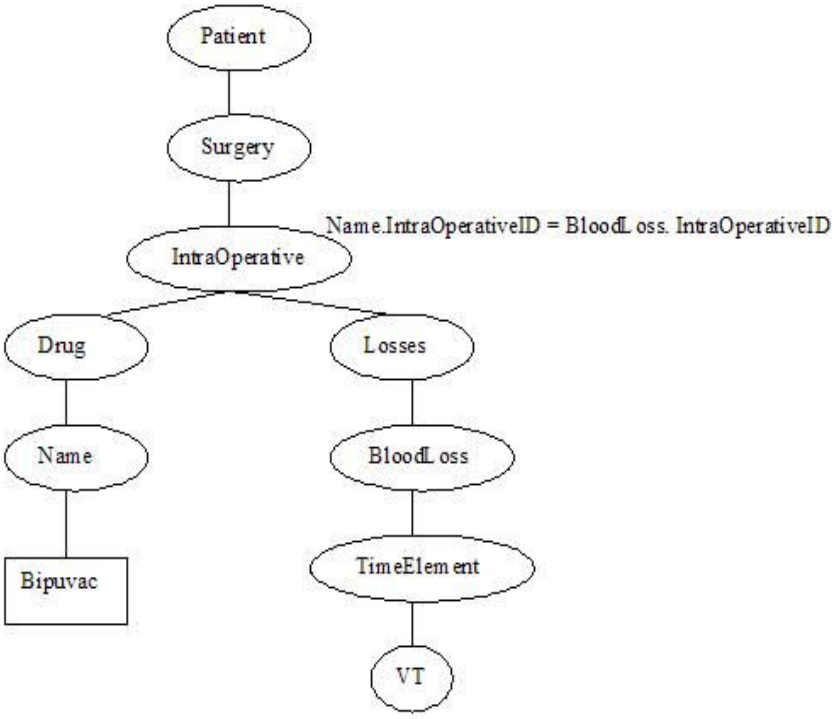


Figure 3.17. Query 3

Temporal Slicing

Query 4: List the surgery names that are valid during 15.00-17.00?

XPath expression of the query: /Patient/Surgery/SurgeryName/TimeElement/VT [low ≤15.00 and high≥17.00]. Figure 3.18 illustrates the Query 4 on the node tree.

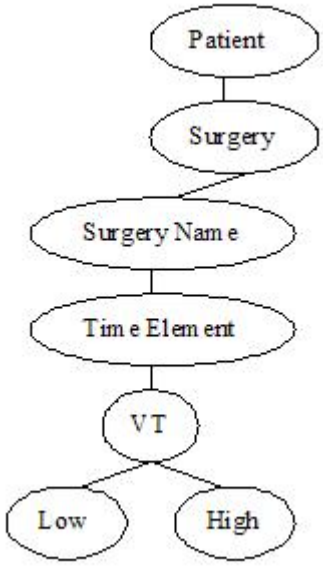


Figure 3.18. Query 4

In order to apply temporal slicing operation on temporal XML document, we use TAR* trees of the query path. In this case, the query has a constraint on valid time dimension. Without accessing the irrelevant time dimensions indexed in TAR* tree we can efficiently return the results of surgery from the "Patient/Surgery/SurgeryName" TAR* table.

Temporal Join

Query 5: What are the drugs that are given to the patients because of a paresthesia complication?

XPath expression of the query:

//patient/surgery/intra-operative//Drug[ancestor::*/Complication[Name=”Paresthesia”]//VT=ET]/Name. Figure 3.19 illustrates the Query 5 on the node tree.

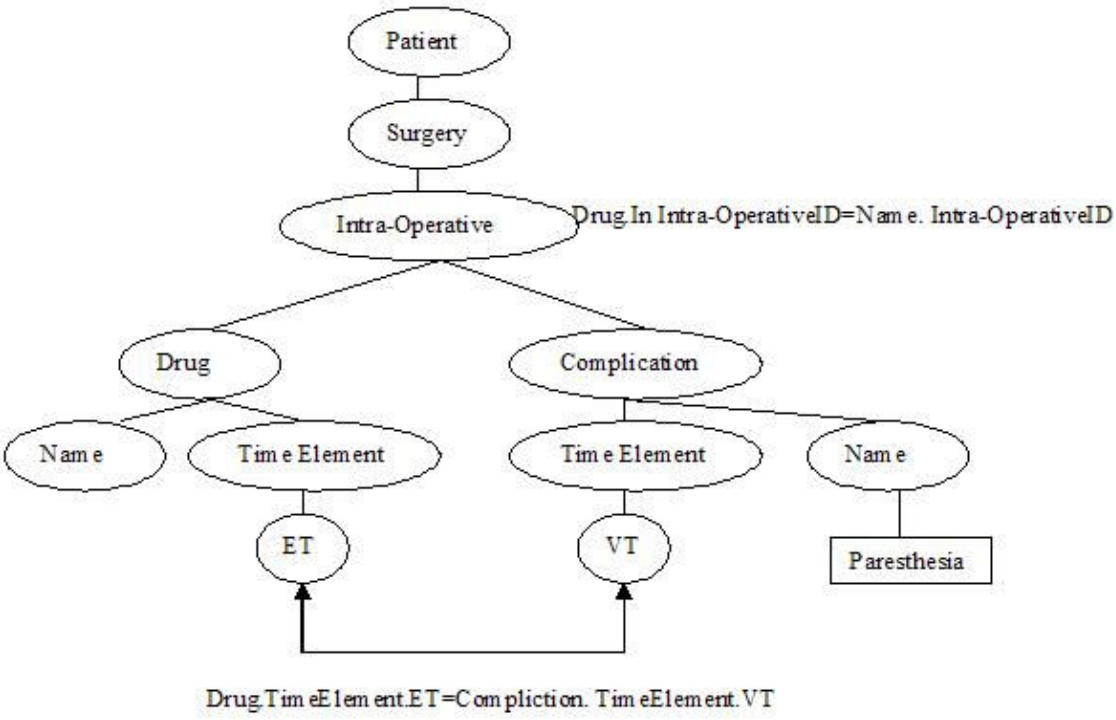


Figure 3.19. Query 5

Using the B+ value index in the “Patient/Surgery/Intra-operative/Complication/Name” path index table we access the paresthesia elements, their enclosing minimum bounding boxes and their ancestors whose element type is Intra-operative. When we find these intra-operative periods in the join table we return the descendants whose path type is “Patient/Surgery/Intra-operative/Drug/Name”. By using the name path table of the drug, we get the enclosing minimum bounding boxes of drug. In order to apply time constraint on nodes, we use MBR relation tables. Overlapping minimum bounding boxes of TAR* index trees are precomputed and stored in the MBR relation tables. We have matched complication nodes and their enclosing MBRs and drug element list and their enclosing MBRs. Instead of comparing each MBR tuple in both lists, we use MBR relation tables to return the drug elements whose minimum bounding box overlaps with the complication list minimum bounding boxes.

Period Containment

Query 6: Who are the patients that have the same surgery history with patient X?

The query execution starts with extracting the patient node id and their surgery element identifiers of the patient X with the help of B+ trees in the "/Patient/Name" table. From the join table we retrieve the minimum bounding regions of the matched surgery element. The final step is retrieving the patients who have the same MBR list with the extracted MBR list in the previous step.

Comparison Queries

Query 7: Find the complications that appeared no more than 30 minutes after their initiating event?
XPath expression of the query is:
//patient/surgery/intra-operative/complication/TimeElement[VT/low – ET/low ≤ 30].
This query is asked for the complication elements whose valid time start point is 30 minutes grater than their initiating event time. The query is executed on the VT_S-ET_S B+ tree. The matched nodes are returned if their element type is complication.

Query 8: Find all pre-operation lab results that are inserted retroactively?
XPath expression of the query is:
//patient/surgery/preoperative/labResults/TimeElement[TT/low – VT/low > 0].
This query is asked for the pre-operation lab elements whose transaction time start point is greater than the valid time start point. The query is executed on the B+ tree of TT_S-VT_S tuple. The matched nodes are returned if their element type is preOperativeLab.

If we don't use B + trees for comparison queries, we have to traverse all elements of the given element type in the query and make a comparison between their start or end time points for each element.

Temporal Partially Dimensional Range Queries

This type of queries may have constraints on the varying combinations of four time dimensions. The combination of four different time dimensions will produce in 2^4-1 query types. Using the corresponding temporal AR* tree of the given path, all combinations can be efficiently queried. Queries 9 and 10 are two examples for temporal partially dimensional range queries and other combinations can be processed in a similar way.

Query 9: List all pre-operative lab results whose availability time contains 01.12.2006 15.00 and 01.12.2006 17.00?

XPath expression of the query is:
/patient/surgery/pre-operative/labresults/TimeElement/AT[Low ≥ 01.12.2006 15.00 and High ≤ 01.12.2006 15.00]
The query is executed on the "/patient/surgery/pre-operative/labresults" TAR* tree, using the availability time nodes.

Query 10: List all pre-operative lab results whose availability time is 01.12.2006 15.00 and transaction time is 01.12.2006 17.00?

XPath expression of the query is:
/patient/surgery/pre-operative/labresults/TimeElement [AT/Low = 01.12.2006 15.00 and TT/Low=01.12.2006 17.00]
The query is executed on the "/patient/surgery/pre-operative/labresults" TAR* tree, using the availability time and transaction time nodes.

4. PERFORMANCE STUDY AND EVALUATIONS

In this chapter, we present the detailed performance study results of the proposed Temporal XML model. The proposed temporal data model contains path indices and a temporal multidimensional index used for temporal queries. The proposed model is compared with two different multidimensional indexing techniques. The first one is Double R tree methodology which is designed for now relative bitemporal data. The second compared structure is AR Tree, which is designed for partially dimensional range queries. While implementing the AR Tree, we use maximum timestamp approach for now relative timestamps.

The proposed model and the comparison methods are implemented in Java. All the experiments are tested on a Intel(R) Pentium(R) M processor 1.80GHz PC with 1GB RAM running Windows XP. The anaesthesia data sets are generated from a sample anaesthesia record. We conducted our experiments using randomly generated dataset of 2100 anaesthesia documents. The anaesthesia documents are composed of three periods: pre-operative, intra-operative and post-operative periods. In the documents, there are 76 different root to leaf paths. In the intra-operative period, anaesthesia data is recorded in every five minutes. So we use the anaesthesia datasets in which the time granularity of the anaesthesia data is in “minutes”.

Although we are interested on temporal queries, we also tested non-temporal queries in the experiments. For the non-temporal queries, we use Selection queries. We tested Temporal Slicing and Temporal Partially Dimensional Range query types as the temporal query types. In order to measure the advantage of the proposed structure over Double R tree methodology, we use partially dimensional range queries with one, two, three and four dimensions. We also searched both logically deleted and alive records in the database for measuring the advantage of the proposed model over maximum timestamp approach that is implemented AR tree model.

Seven different types of queries are tested on the Double R Tree structure, AR Tree structure and the proposed model. The measurements are done in terms of query processing time. Table 4.1 shows the query types that were used during experiments.

Table 4.1. Query Types

	Query Type	Query
1	Temporal Slicing	List the drug amounts that are valid during 15.00-17.00?
2	Temporal Partially Dimensional Range Query (2 Dimension)	List all pre-operative lab results of wbc whose availability time contains 12.10.2006 16.00 and 12.10.2006 20.30 and valid time contains 12.10.2006 16.00 and 12.10.2006 20.30?
3	Temporal Partially Dimensional Range Query (3 Dimension)	List all pre-operative lab results of potassium whose availability time contains 12.10.2006 16.00 and 12.10.2006 20.30 and valid time contains 12.10.2006 16.00 and 12.10.2006 20.30 and transaction time contains 12.10.2006 16.00 and 12.10.2006 20.30?
4	Temporal Fully Dimensional Range Query (4 Dimension)	List all pre-operative lab results of glucose whose availability time contains 12.10.2006 16.00 and 12.10.2006 20.30 and valid time contains 12.10.2006 16.00 and 12.10.2006 20.30 and transaction time contains 12.10.2006 16.00 and 12.10.2006 20.30 and event time contains 12.10.2006 16.00 and 12.10.2006 16.00?
5	Temporal Partially Dimensional Range Query (1 Dimension – [time, time])	List all pre-operative lab results of creatinine whose transaction time contains 12.10.2006 16.00 and 12.10.2006 21.30?
6	Temporal Partially Dimensional Range Query (Transaction Time)	List all pre-operative lab results of wbc whose transaction time contains 12.10.2006 16.00
7	Selection	List the surgeries of the surgeon “J. Smith”.

All queries are tested for ten times and the execution times in the figures 4.1-4.6 show the mean of the results. Figure 4.8 shows the performance of the proposed model over AR model and 4.9 shows the performance of the proposed model over Double R model.

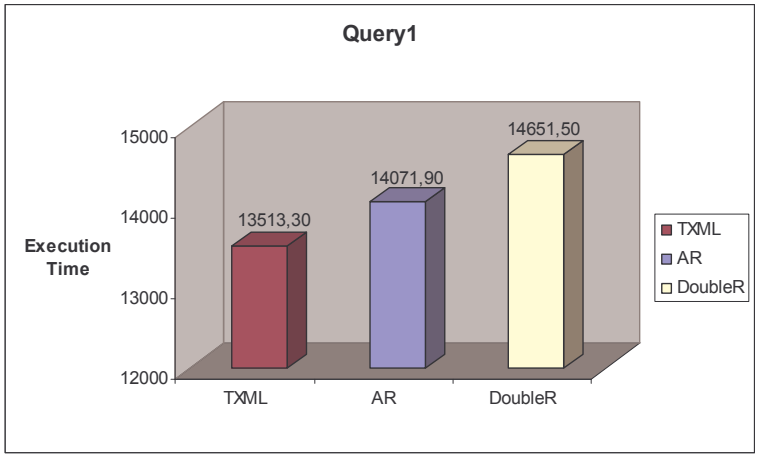


Figure 4.1. Execution time of query 1 (in ms)

Figure 4.1 shows the execution time of the query 1 in terms of milliseconds. Since there is only one time dimension in the query, we expect shorter execution times for the proposed model than Double R model which is designed for fully dimensional queries. In the proposed model, valid time and event time dimensions are bounded, i.e. they don't contain now relative timestamps. In AR tree, now relative time stamps are inserted using maximum timestamp approach and bounded timestamps are inserted according to their values. As we expected, query 1 shows 7.77 per cent better performance than Double R model and similar performance, 3.97 percent, with AR model because of the bounded time dimension.

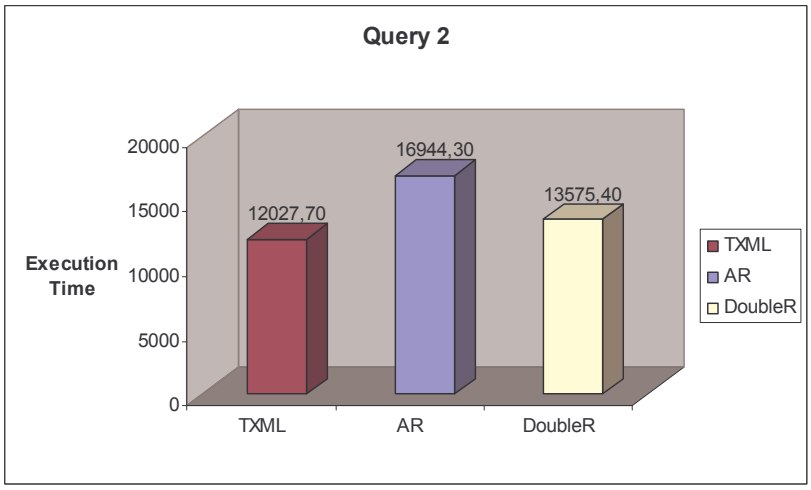


Figure 4.2. Execution time of query 2 (in ms)

Figure 4.2 shows the execution time of the query 2 in terms of milliseconds. Query 2 is a temporal partially dimensional range query with two time dimensions. Since the number of query dimensions is smaller than the data dimensions, the proposed model should have shorter execution time than Double R model. The query is asked for the valid time and availability time which contains now relative time data. So we expect better performance over AR model because of the now relative data. As we expected, the proposed model is 29.02 per cent faster than AR model and 11.4 per cent faster than Double R model.

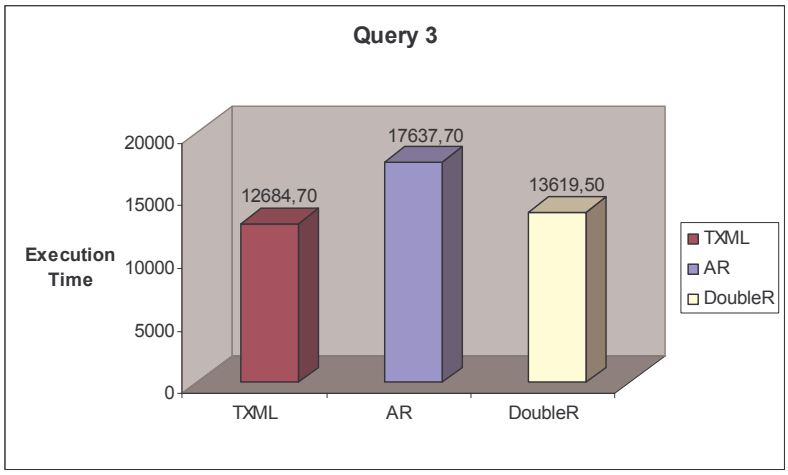


Figure 4.3. Execution time of query 3 (in ms)

Figure 4.3 shows the execution time of the query 3 in terms of milliseconds. Query 3 is a temporal partially dimensional range query with three time dimensions. Since the number of query dimensions is smaller than the data dimensions, the proposed model should have shorter execution times than Double R model. However, as the number of query dimensions becomes closer to the number of data dimensions, we expect the performance of the proposed model will come closer to Double R model. The query is asked for the valid time, transaction time and availability time which contain now relative time data. So we expect better performance over AR model because of the now relative data in transaction time and availability time dimensions. As we expected, the proposed model is 28.08 per cent faster than AR model and 6.86 per cent than Double R model. If we compare the performance of proposed model over Double R model in query 2 and 3, we can see the performance degradation for higher number of query dimensions.

Figure 4.4 shows the execution time of the query 4 in terms of milliseconds. Query 4 is a temporal fully dimensional range query with four time dimensions. Since the number of query dimensions is equal to the number of data dimensions, we expect similar performance with Double R model. We also expect better performance over AR model because of now relative data. In the executions, TXML shows 27.64 per cent better performance than AR model and only 3.79 per cent better performance than Double R model as we expected.

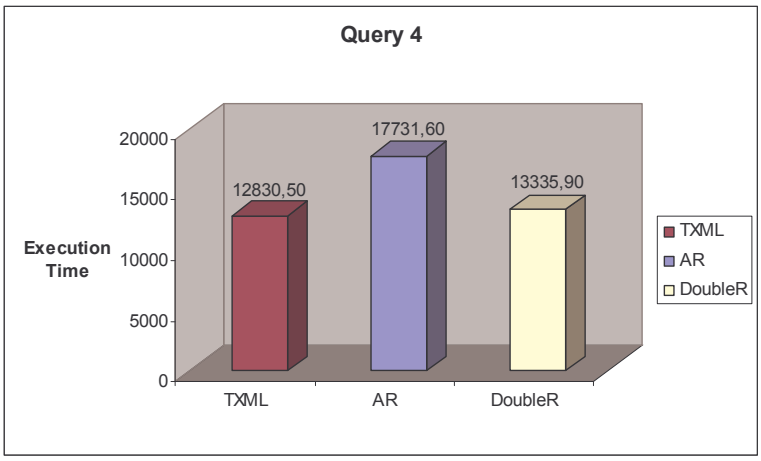


Figure 4.4. Execution time of query 4 (in ms)

Figure 4.5 shows the execution time of the query 5 in terms of milliseconds. Query 5 is a temporal partially dimensional range query with one time dimension. The query is asked for the transaction time which contains now relative time data. Query 1 is similar with query 5 since they both query one dimension so we expect similar results. However, in query 5 the performance is higher, so we can conclude that TXML shows better performance for now relative data. In the executions, TXML shows 28.71 per cent better performance than AR model and only 15.37 per cent better performance than Double R model as we expected.

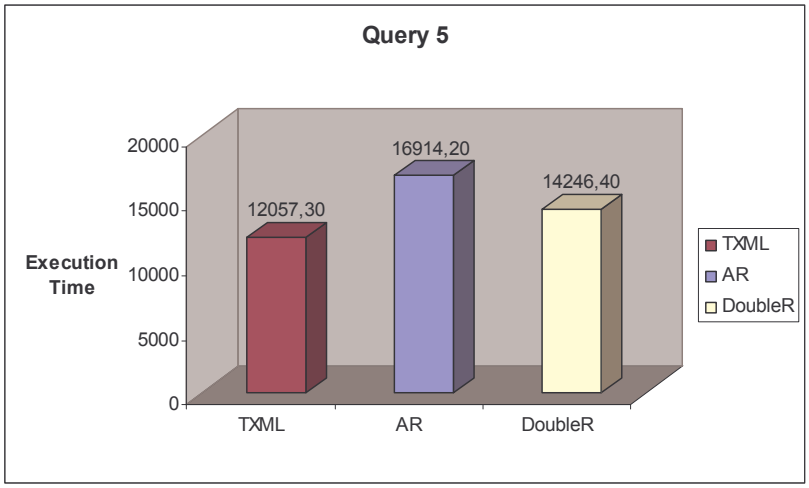


Figure 4.5. Execution time of query 5 (in ms)

Figure 4.6 shows the execution time of the query 6 in terms of milliseconds. Query 6 is a temporal partially dimensional point query with one time dimension. The query is asked for the transaction time which contains now relative time data. As in the one dimensional transaction time range query, point query shows better performance, 29.73 per cent, than AR model and 13.09 per cent better performance than Double R model as we expected.

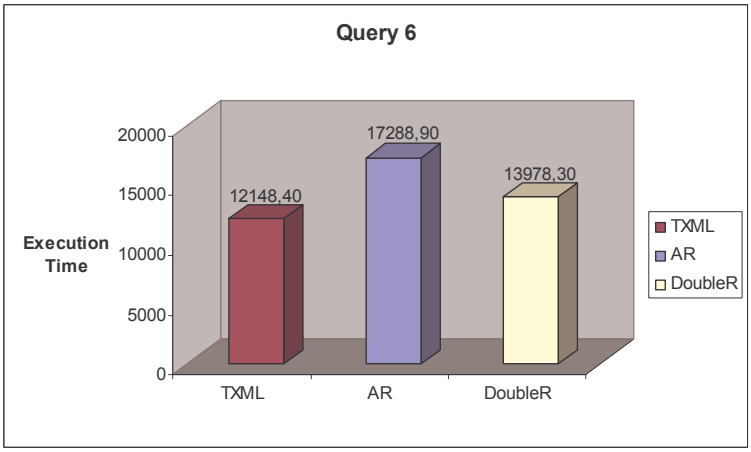


Figure 4.6. Execution time of query 6 (in ms)

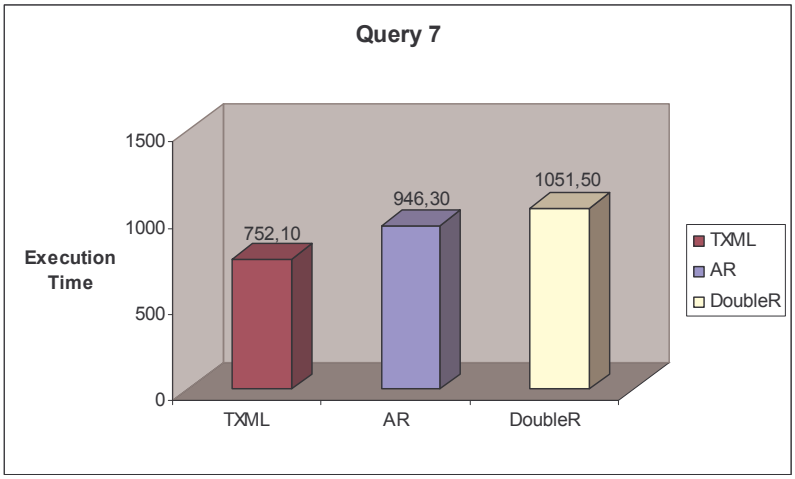


Figure 4.7. Execution time of query 7 (in ms)

Figure 4.7 shows the execution time of the query 7 in terms of milliseconds. Query 7 is a non-temporal selection query. Although three models use same structures to answer the query, the structures contain time data and it affects the performance of the non-temporal queries. In the executions, the proposed model 20.52 per cent, than AR model and 28.47 per cent better performance than Double R model.

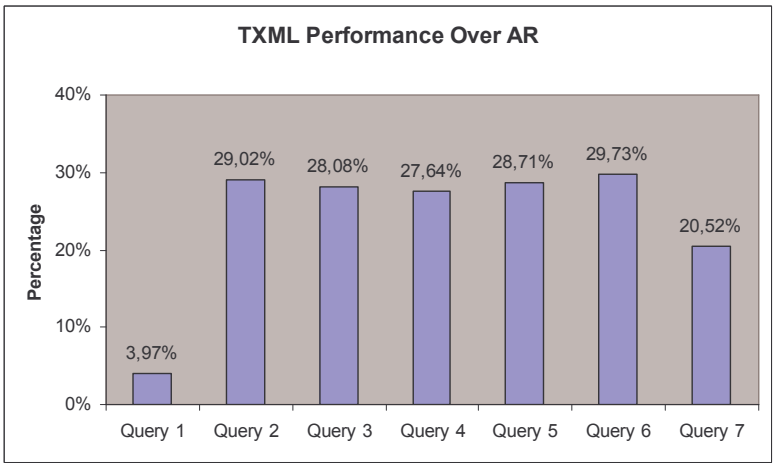


Figure 4.8. Performance of proposed model over AR model

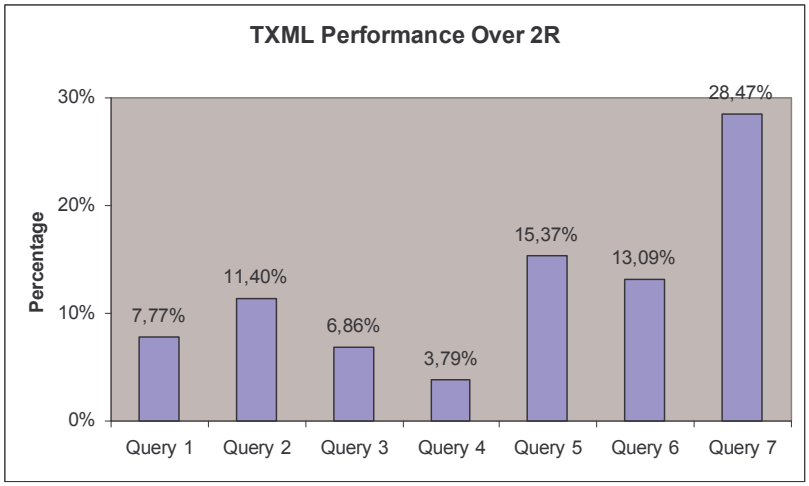


Figure 4.9. Performance of proposed model over Double R model

The experiments show that the proposed model has better performance than AR model. When we compare the performance of the proposed model over AR model for now relative timestamps and bounded timestamps, the experiments show that for now relative timestamps the proposed model has better performance than bounded timestamps. The performance of the proposed model is better than Double R model in all experiments, but as the number of query dimensions decreases, the proposed model shows better performance.

5. CONCLUSION

The proposed model, “Temporal XML Data Model for Anaesthesia Data”, has three main contributions. These can be summarized as, it attaches the time information to the anaesthesia data, and records four different time dimensions of anesthesia data and it efficiently processes the partially dimensional temporal queries.

The proposed data model uses XML so that it solves the problem of data exchange and different structures in health information systems. Time information of the anesthesia data is important because of research, audit and medico-legal issues. Most of the current temporal XML models only records valid time and transaction time. However decision and availability times are also needed in order to store and represent information accurately. The proposed model records valid time, transaction time, event time and availability time, in order to provide the temporal information of the events during anaesthesia.

The proposed data model consists of index structures to decrease frequent disk accesses which improves the performance of the system. Non-temporal queries can be efficiently processed by the path index and join index structures without traversing the anaesthesia document tree. In order to process temporal queries, a temporal index structure, TAR Index, is proposed. TAR Index uses the advantage of AR Tree over partially dimensional range queries. It stores the alive and logically deleted records in two different trees and avoids large rectangles which will lead excessive dead spaces and overlaps for now-relative data.

We compare the TXML with AR model and Double R model in the experiments. For partially dimensional queries, our proposed model shows better performance than Double R model. As a second result, our proposed model shows better performance over AR model for now relative time queries.

In conclusion, the proposed model combines the requirements of time-related data and XML. In order to store information accurately, the model attaches four time

dimensions to the data. Generally, most of the queries don't contain all data dimensions at the same time. Therefore, using the proposed time index structure, temporal partially dimensional range queries can be processed efficiently. By recording the four temporal time dimensions of anaesthesia data in the proposed model, many questions can be answered about medico legal issues in some critical situations, illegal modifications on the records can be easily detected, malpractices in anesthesia information systems can be exposed and the barrier for widespread acceptance of AIMS technology can be eliminated.

APPENDIX A: XML: Extensible Markup Language

Extensible Markup Language (XML) describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. The main purpose of XML is to mark up documents. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure [16].

Each XML document has a physical structure. The document is composed of entities, which may refer to other entities to cause their inclusion in the document. An XML document begins in a "root" or document entity and the structure of the document must be nested properly.

XML documents contain one or more element entities, which are either delimited by start-tags and end-tags, or by an empty-element tag. An XML element is everything from the element's start tag to the element's end tag. Each element has a type and a name, which identifies the element. An element can have element content, mixed content, simple content, or empty content. Elements in XML documents are related as parents and children. If an element contains subelements, than it is the parent of the subelements.

In addition to the sub elements, an element may contain a set of attributes, which provide additional information about elements. Each attribute specification has a unique name (inside the element) and a value. XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The string type may take any literal string as a value; the tokenized types are more constrained.

Data can be stored in sub elements or in attributes. There are no rules to use a subelement or an attribute. However, there are some differences between elements and attributes.

Some of the differences are:

- Attributes are defined uniquely in elements and cannot contain markup and are thus guaranteed to be atomic.
- Elements can occur more than once within the same level, while attributes can only appear once within the same level.
- Elements can be defined to be in a certain order, while attributes can appear in any order.
- Attributes cannot contain multiple values while subelements elements can.

XML is an industry-standard, system-independent way of representing data. It is designed to describe data and to focus on what data is. Because XML tags indicate the content and structure of the data they enclose, they make it possible to do things like archiving and searching. The hierarchical structure of the XML makes possible to naturally represent the history of data.

Figure A.1 shows a list of patients and their doctors. <patientList> is the root element in the figure and it consists of child <patient> elements. The <patient> and </ patient > tags tell a parser that the information between them is about a patient. The child tags inside the < patient > tags specify that the enclosed information is the patient's surgeon's and anesthetist's name.

```
<patientList>
    <patient>
        <name>John Brown</name>
        <surgeonName>Dr. Jim Williams </surgeonName>
        <anaesthesistName>Dr. Mary Wiley</anaesthesistName>

    </patient>
    <patient>
        <name>James Carey</name>
        <surgeonName> Dr. Jim Williams </surgeonName>
        <anaesthesistName>Dr. Jill Jones</anaesthesistName>
    </patient>
</patientList>
```

Figure A.1. Sample XML Document

The structure of the XML documents can be described with a schema file. A well-formed XML document is a document that conforms to the XML syntax rules and a schema.

There are two different schema languages; Document Type Definition (DTD) and XML Schema. A schema written in Document Type Definition language is commonly referred to as a DTD. The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD consists of a set of production rules for elements (tags) that have a name and describe its content as empty, any, mixed, choice or sequence. An element can also contain attributes that are declared separately.

While DTDs are appropriate for marking up text, they are very limited for other applications. Consequently, a new schema language was developed by the World Wide Web (W3C) consortium. This new schema language called as XML Schema [17] [18] [19]. The XML Schema language is also referred to as XML Schema Definition (XSD).

An XML schema defines the elements and attributes that can appear in the XML documents. It defines the parent child relationships, the order and the number of the elements. Context type of the elements and default values are defined in XML schema. The most important fact is that XML Schemas are very flexible and allow to describe the same rules in many different ways depending on the use of the structuring concepts such as primitive data types including byte, date, integer, string, simple and complex types, type inheritance, restrictions and extensions, global and local definitions and embedded, flat catalog and named type structuring constructs.

The differences between the XML Schema and DTD are; XML Schemas are extensible to future additions and more powerful than DTDs. They are written in XML and support data types and namespaces.

The XML Schema of the XML document in Figure A.1 is displayed in Figure A.2. The <element name="patientList"> line defines the document element named `patientList` of type complex meaning it includes other tags. Element `patient` is also a complex type and includes a sequence of three tags named `name, surgeonName` and `anaesthetistName` of type string.

The minOccurs and maxOccurs specify that a patient element can occur zero or more times inside the `patientList` element.

The XML Schema provides very rich constructions for specifying the structure and content of documented information of almost any complexity. In addition, there are powerful standards and tools which support display, filtering, transformation or linkage of information, and its retrieval using queries.

```
<schema>
   <element name="patientList">
      <complexType>
            <element name="patient" minOccurs=”0” maxOccurs=”unbounded”>
            <complexType >
             <sequence>
               <element name="name" type=”string”/>
               <element name="surgeonName" type=”string”/>
               <element name="anaesthetistName" type="string"/>
             </sequence>
            </complexType>
            </element>
      </complexType>
   </element>
```

Figure A.2. Sample XML Schema Definition

APPENDIX B: XML DOM

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document." [20]. The W3C DOM provides a standard set of objects for HTML and XML documents, and a standard interface for accessing and manipulating them.

The XML Document Object Model (XML DOM) defines a standard way for accessing and manipulating XML documents. The DOM presents an XML document as a tree-structure with the elements, attributes, and text defined as nodes. The entire document is the document node. Every XML tag is an element node. The text values of the elements are assigned to text nodes. Every XML attribute is an attribute node and comment elements are comment nodes.

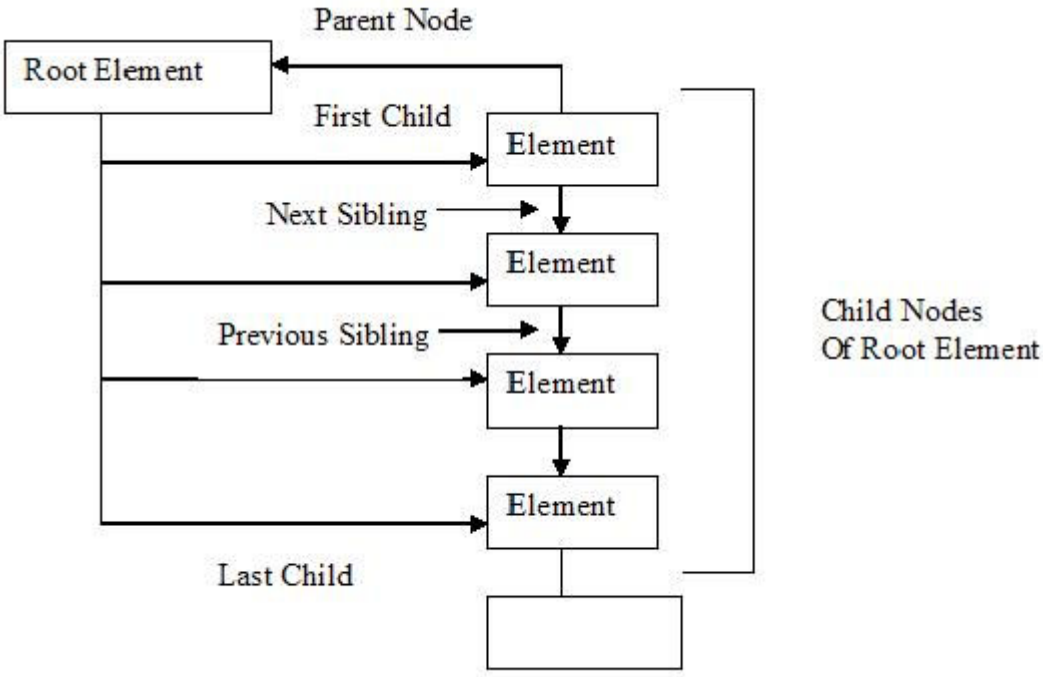


Figure B.1. Relationship between nodes in the XML file.

A node tree in Figure B.1 shows an XML document as a set of nodes and the connections between them. In a node tree, the top node is called the root. Every node

except root has exactly one parent node. If an element with subelements is defined in an XML document than it is denoted as a parent node in the node tree. The subelements are the child nodes, which are also sibling nodes in the tree.

To manipulate an XML document, an XML parser is needed. Parsing is the process of reading continuous stream of characters and construction of meaningful tokens that could be processed by computer programs. There are two methods for parsing XML documents, the Document Object Model (DOM) and the Simple API for XML (SAX) [21] [22].

DOM creates a tree form representation of an XML document. In the first pass it reads the entire document and stores in memory. Then it constructs the DOM tree model of the document. In the second pass, the application accesses the data that is parsed completely and stored in memory. For applications that include large documents and involve one-time read/write per parse, DOM presents a considerable overhead on memory. DOM is best used for applications where the document elements have to be randomly accessed and manipulated.

SAX parsers handle XML documents as a stream and work unidirectional, which means that they cannot return back to previous nodes without reparsing the document. XML constructs are pushed to the application by the parser in the form of callbacks. The programmer provides callback methods and the parser invokes these methods as part of traversal of the XML document. They also require at least two passes of data, one for identification of elements and one by the application. Since SAX parsers work in stream mode they are faster than DOM requiring smaller memory.

APPENDIX C: XPATH

XPath is a stand-alone language for finding information in an XML document and also a part of XSLT and XQuery. XPath is used to navigate through elements and attributes in an XML document. It uses path expressions to select nodes or node-sets in an XML document.

The W3C XPath recommendation does not provide a formal model [23]. However, the XPath data model is commonly assumed to be an ordered tree. The tree represents the nesting of elements within the document, with elements corresponding to nodes, and element content are represented by text nodes, which are child nodes of element nodes. Unlike a tree, the children for a node are ordered based on their physical position within the document. In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document (root) nodes.

The main components of an XML documents in XPath data model are the root, element, text, and attribute nodes. The root node is the root of the tree. A root node can only be in the root of the tree. The element node for the document element is a unique child of the root node. There is an element node for every element in the document. The children of an element node are element and/or text nodes. Each element node has an associated set of attribute nodes. The contents of the simple element nodes or attribute nodes can be represented by a text node. For every type of node, we can determine a string-value for a node of that type. Figure C.1 displays a sample XPath data model.

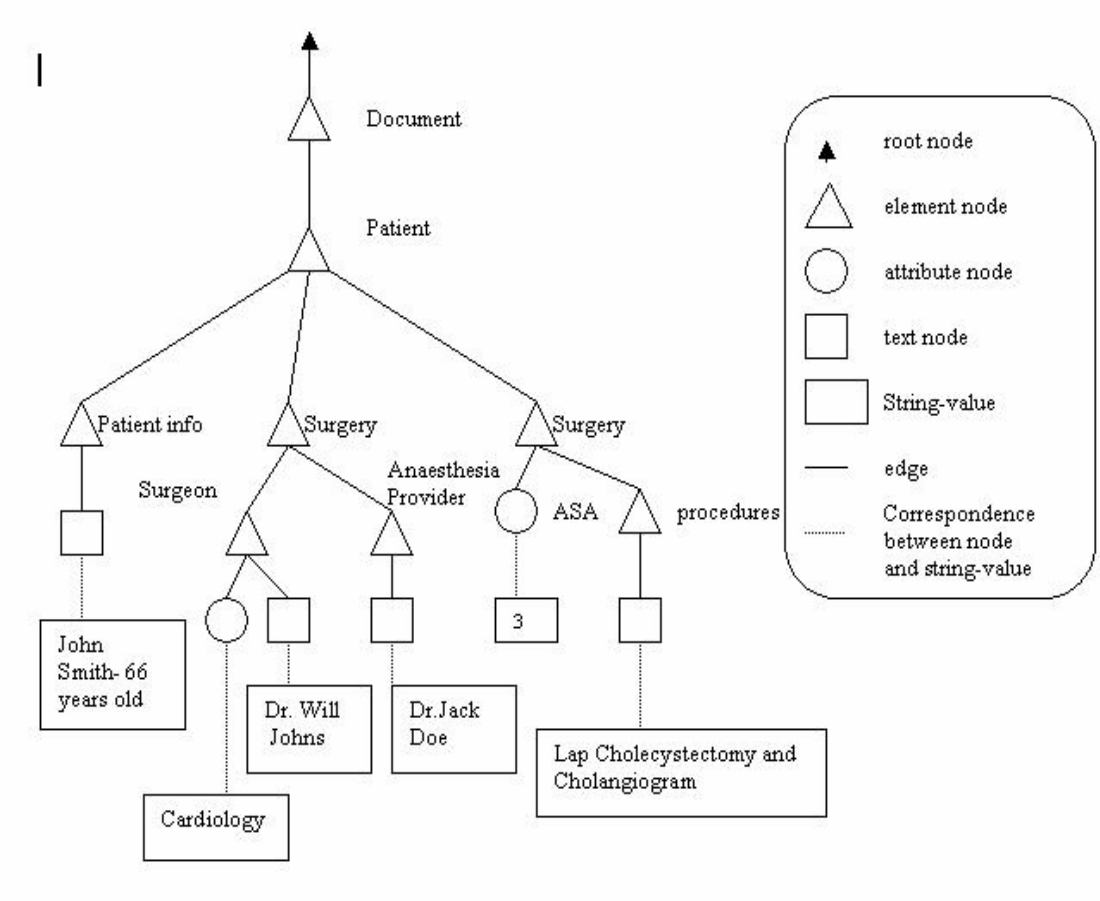


Figure C.1. Relationship between nodes in the XML file.

An XPath expression consists of a sequence of steps. A step of an XPath expression can have a context, an axis, a node test and some predicates. The context node is the starting point of the evaluation. The axis specifies the relationship between the nodes selected by the axis step and the context node. There are 13 axes defined in XPath: ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self, following, preceding, following-sibling, preceding-sibling, namespace, parent and Self. The node test specifies the node type of the nodes selected by the axis step. The symbol "::" is used to separate a node test from an axis. A predicate, enclosed in square brackets, filters a node sequence so that retain some nodes and discard others. There can be zero or more predicates in one step. The nodes selected after the axis and the node test are finally evaluated against the predicates in the order that they appear in the step [24]. The syntax of an XPath expression: axis specifier :: node test [predicate]*

XPath expression is used for finding paths in the XML document. One of the frequently used XPath expressions is location path. A location path identifies a set of nodes in a document. The simplest location path is the one that selects the root node of the document. This is simply the forward slash /. When a root location path is used, it retrieves all the nodes under the root node in the tree. The second simplest location path is a single element name. This path selects all child elements of the context node with the specified name. Attributes are also selected by using XPath expressions. In order to select an attribute, @ sign followed by the name of the attribute is used.

Parent-Child and ancestor-descendant relationship can be extracted from the document tree using XPath expressions. The single slash (“/”) selects the child nodes of the given element. The double slash (“//”) selects all the descendants of the context node as well as the context node itself.

In general an XPath expression retrieves more than one node. In order to minimize the result set, some predicates can be used. XPath provides arithmetic operators for addition, subtraction, multiplication, division, and modulus, in their usual binary and unary forms. Comparison expressions allow two values to be compared. XPath provides three kinds of comparison expressions, called value comparisons, general comparisons, and node comparisons. XPath supports a full complement of relational operators, including =, <, >, >=, <=, and !=.

XPath provides core functions that are useful in predicates or raw expressions. Function call expression is evaluated by using the function name to identify a function in the expression evaluation context function library, evaluating each of the arguments, converting each argument to the type required by the function, and finally calling the function, passing it the converted arguments. Every implementation of XPath must implement these functions. Each XPath function returns one of these four types: Boolean, Number, Node-set, String. A node-set function takes a node-set argument, returns a node-set, or returns/provides information about a particular node within a node-set. String functions perform evaluations, formatting, and manipulation on string arguments. Boolean and number functions evaluate the argument expressions to obtain a Boolean or numeric result. The figure C.2 shows some examples of XPath expressions.

/patient = Selects the patient elements

* = Matches any element

/ = Matches the root element

/patientList = Matches the patientList element under the root

/patientList/patient = Matches the patient elements under the patientList element

/patientList//surgeonName = Matches the surgeonName elements under the patientList.

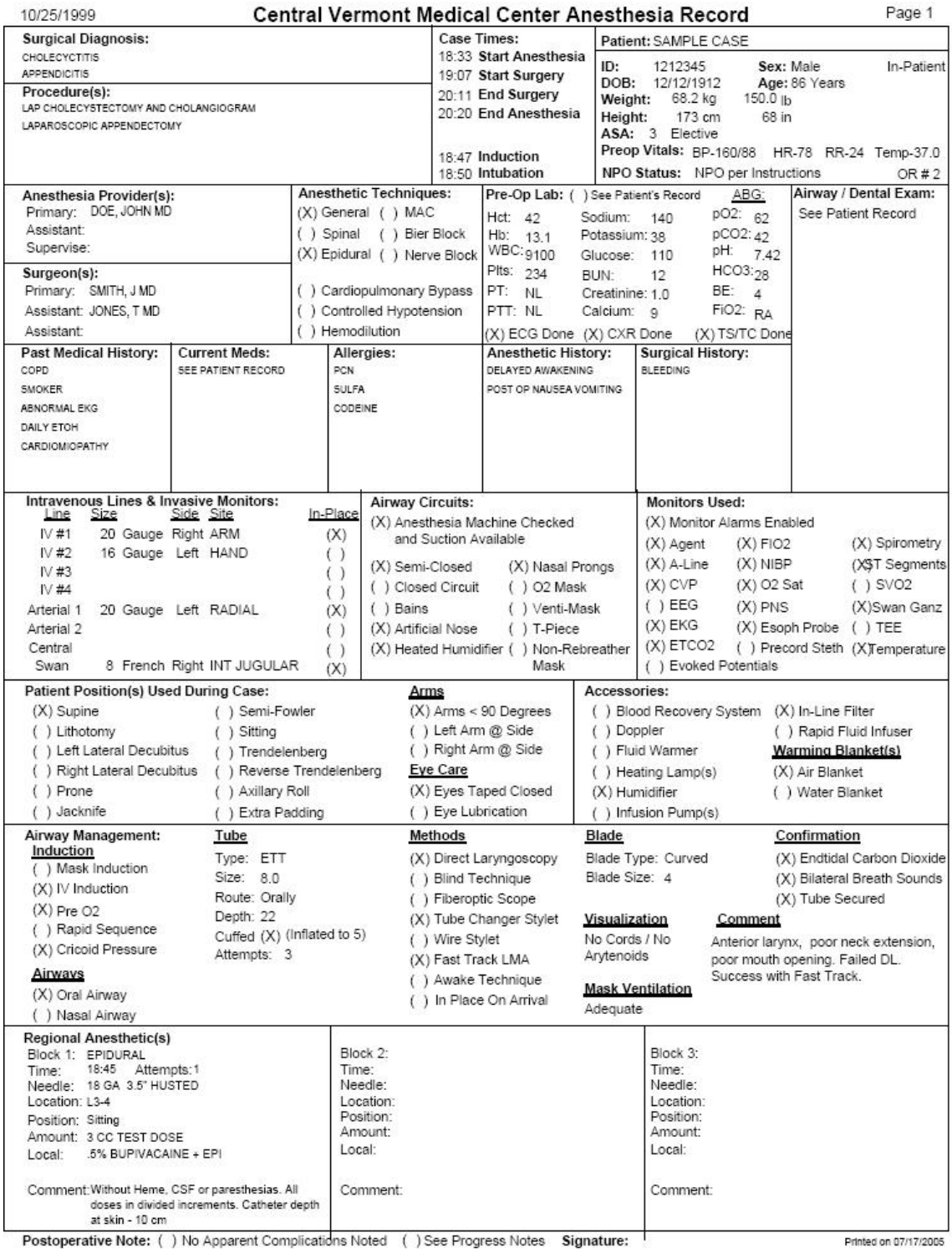
//anaesthetistName = Matches the anaesthetist elements in any depth

//patient[name=" John Brown"] = Matches the patient elements whose name is "John Brown"

Figure C.2. XPath Expression examples

APPENDIX D: SAMPLE ANAESTHESIA RECORD

Currently SIGGAS, the HL7 special interest group of generation of anaesthesia standards, is working for creating HL7 CDA-compliant anaesthesia schemas in order to develop anesthetic specificity in data standards and augment the HL7 model for quality improvement in anesthetic patient care. The anaesthesia records that are used in anaesthesia information systems in the healthcare instutitions do not have a standard but have a similar structure. In figure D.1 and D.2 we present a sample anaesthesia record [57].



10/25/1999 **Central Vermont Medical Center Anesthesia Record** Page 1

Surgical Diagnosis:
CHOLECYCTITIS
APPENDICITIS

Procedure(s):
LAP CHOLECYSTECTOMY AND CHOLANGIOGRAM
LAPAROSCOPIC APPENDECTOMY

Case Times:
18:33 **Start Anesthesia**
19:07 **Start Surgery**
20:11 **End Surgery**
20:20 **End Anesthesia**
18:47 **Induction**
18:50 **Intubation**

Patient: SAMPLE CASE
ID: 1212345 **Sex:** Male In-Patient
DOB: 12/12/1912 **Age:** 86 Years
Weight: 68.2 kg 150.0 lb
Height: 173 cm 68 in
ASA: 3 Elective
Preop Vitals: BP-160/88 HR-78 RR-24 Temp-37.0
NPO Status: NPO per Instructions OR # 2

Anesthesia Provider(s):
Primary: DOE, JOHN MD
Assistant:
Supervise:

Surgeon(s):
Primary: SMITH, J MD
Assistant: JONES, T MD
Assistant:

Anesthetic Techniques:
(X) General ( ) MAC
( ) Spinal ( ) Bier Block
(X) Epidural ( ) Nerve Block
( ) Cardiopulmonary Bypass
( ) Controlled Hypotension
( ) Hemodilution

Pre-Op Lab: ( ) See Patient's Record

				ABG:	
Hct:	42	Sodium:	140	pO2:	62
Hb:	13.1	Potassium:	38	pCO2:	42
WBC:	9100	Glucose:	110	pH:	7.42
Plts:	234	BUN:	12	HCO3:	28
PT:	NL	Creatinine:	1.0	BE:	4
PTT:	NL	Calcium:	9	FiO2:	RA

(X) ECG Done (X) CXR Done (X) TS/TC Done

Airway / Dental Exam:
See Patient Record

Past Medical History:
COPD
SMOKER
ABNORMAL EKG
DAILY ETOH
CARDIOMIOPATHY

Current Meds:
SEE PATIENT RECORD

Allergies:
PCN
SULFA
CODEINE

Anesthetic History:
DELAYED AWAKENING
POST OP NAUSEA VOMITING

Surgical History:
BLEEDING

Intravenous Lines & Invasive Monitors:

Line	Size		Side	Site	In-Place
IV #1	20	Gauge	Right	ARM	(X)
IV #2	16	Gauge	Left	HAND	( )
IV #3					( )
IV #4					( )
Arterial 1	20	Gauge	Left	RADIAL	(X)
Arterial 2					( )
Central					( )
Swan	8	French	Right	INT JUGULAR	(X)

Airway Circuits:
(X) Anesthesia Machine Checked and Suction Available
(X) Semi-Closed (X) Nasal Prongs
( ) Closed Circuit ( ) O2 Mask
( ) Bains ( ) Venti-Mask
(X) Artificial Nose ( ) T-Piece
(X) Heated Humidifier ( ) Non-Rebreather Mask

Monitors Used:
(X) Monitor Alarms Enabled
(X) Agent (X) FIO2 (X) Spirometry
(X) A-Line (X) NIBP (X) ST Segments
(X) CVP (X) O2 Sat ( ) SVO2
( ) EEG (X) PNS (X) Swan Ganz
(X) EKG (X) Esoph Probe ( ) TEE
(X) ETCO2 ( ) Precord Steth (X) Temperature
( ) Evoked Potentials

Patient Position(s) Used During Case:
(X) Supine ( ) Semi-Fowler
( ) Lithotomy ( ) Sitting
( ) Left Lateral Decubitus ( ) Trendelenberg
( ) Right Lateral Decubitus ( ) Reverse Trendelenberg
( ) Prone ( ) Axillary Roll
( ) Jacknife ( ) Extra Padding

Arms
(X) Arms < 90 Degrees
( ) Left Arm @ Side
( ) Right Arm @ Side

Eye Care
(X) Eyes Taped Closed
( ) Eye Lubrication

Accessories:
( ) Blood Recovery System (X) In-Line Filter
( ) Doppler ( ) Rapid Fluid Infuser
( ) Fluid Warmer **Warming Blanket(s)**
( ) Heating Lamp(s) (X) Air Blanket
(X) Humidifier ( ) Water Blanket
( ) Infusion Pump(s)

Airway Management:
Induction
( ) Mask Induction
(X) IV Induction
(X) Pre O2
( ) Rapid Sequence
(X) Cricoid Pressure

Airways
(X) Oral Airway
( ) Nasal Airway

Tube
Type: ETT
Size: 8.0
Route: Orally
Depth: 22
Cuffed (X) (Inflated to 5)
Attempts: 3

Methods
(X) Direct Laryngoscopy
( ) Blind Technique
( ) Fiberoptic Scope
(X) Tube Changer Stylet
( ) Wire Stylet
(X) Fast Track LMA
( ) Awake Technique
( ) In Place On Arrival

Blade
Blade Type: Curved
Blade Size: 4

Visualization
No Cords / No Arytenoids

Mask Ventilation
Adequate

Confirmation
(X) Endtidal Carbon Dioxide
(X) Bilateral Breath Sounds
(X) Tube Secured

Comment
Anterior larynx, poor neck extension, poor mouth opening. Failed DL. Success with Fast Track.

Regional Anesthetic(s)

Block 1: EPIDURAL
Time: 18:45 Attempts: 1
Needle: 18 GA 3.5" HUSTED
Location: L3-4
Position: Sitting
Amount: 3 CC TEST DOSE
Local: .5% BUPIVACAINE + EPI
Comment: Without Heme, CSF or paresthesias. All doses in divided increments. Catheter depth at skin - 10 cm

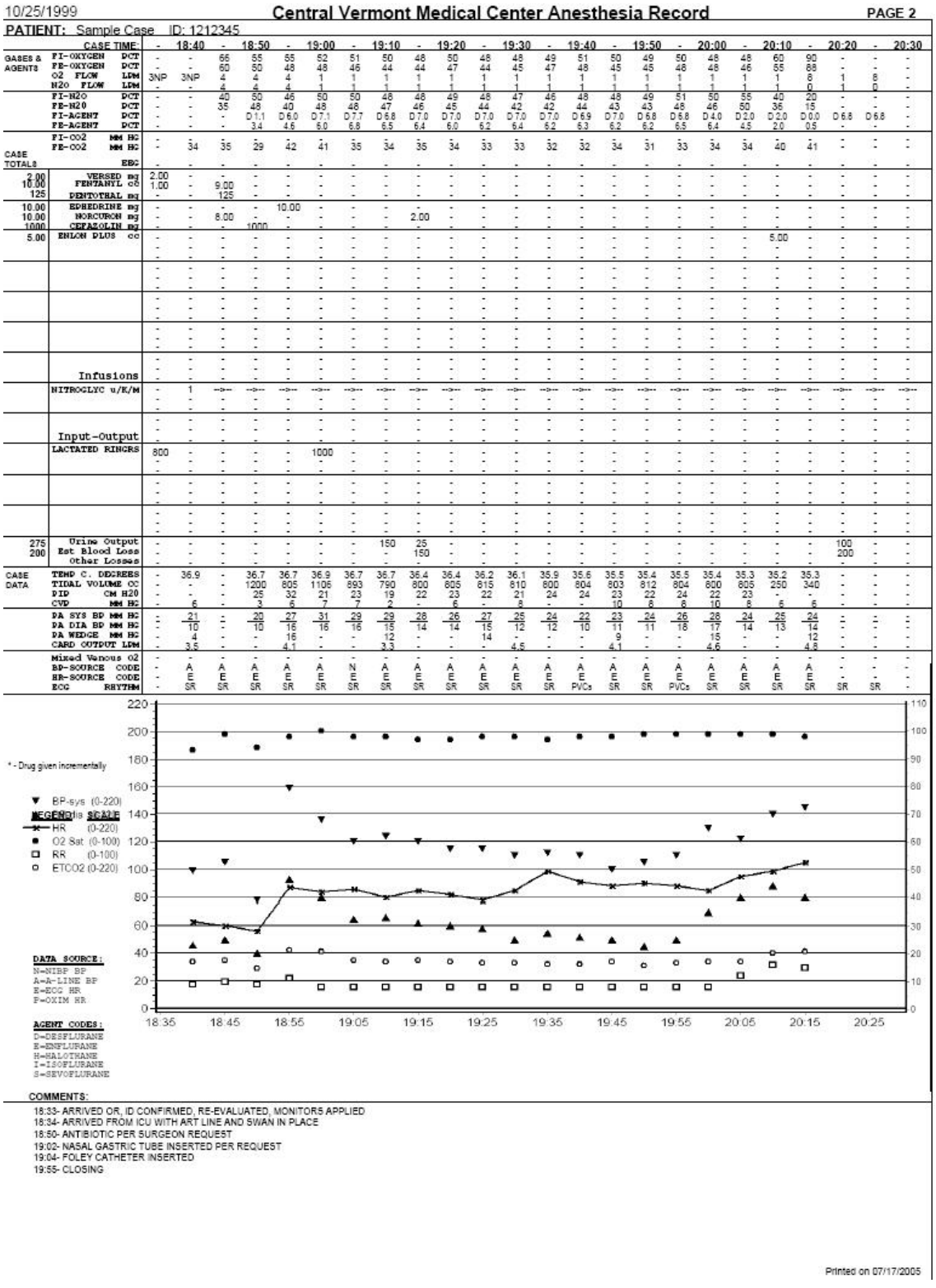
Block 2:
Time:
Needle:
Location:
Position:
Amount:
Local:
Comment:

Block 3:
Time:
Needle:
Location:
Position:
Amount:
Local:
Comment:

Postoperative Note: ( ) No Apparent Complications Noted ( ) See Progress Notes **Signature:**

Printed on 07/17/2005

Figure D.1. Sample Anesthesia Record - Page 1



10/25/1999 **Central Vermont Medical Center Anesthesia Record** PAGE 2

PATIENT: Sample Case ID: 1212345

		CASE TIME	-	18:40	-	18:50	-	19:00	-	19:10	-	19:20	-	19:30	-	19:40	-	19:50	-	20:00	-	20:10	-	20:20	-	20:30
GASES & AGENTS	FI-OXYGEN	PCT	-	-	66	55	55	52	51	50	48	50	48	48	49	51	50	49	50	48	48	60	90	-	-	-
	FE-OXYGEN	PCT	-	-	60	50	48	48	46	44	44	47	44	45	47	48	45	45	48	48	46	55	88	-	-	-
	O2 FLOW	LPM	3NP	3NP	4	4	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	6	1	6	-
	N2O FLOW	LPM	-	-	4	4	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	-
	FI-H2O	PCT	-	-	40	50	46	50	50	48	48	49	48	47	46	48	48	49	51	50	55	40	20	-	-	-
	FE-H2O	PCT	-	-	35	48	40	48	48	47	46	45	44	42	42	44	43	43	48	46	50	35	15	-	-	-
	FI-AGENT	PCT	-	-	-	D 1.1	D 6.0	D 7.1	D 7.7	D 6.8	D 7.0	D 7.0	D 7.0	D 7.0	D 7.0	D 6.9	D 7.0	D 6.8	D 6.8	D 4.0	D 2.0	D 2.0	D 0.0	D 6.8	D 6.8	-
	FE-AGENT	PCT	-	-	-	3.4	4.6	6.0	6.8	6.5	6.4	6.0	6.2	6.4	6.2	6.3	6.2	6.2	6.5	6.4	4.5	2.0	0.5	-	-	-
	FI-CO2	MM HG	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	FE-CO2	MM HG	-	34	35	29	42	41	35	34	35	34	33	33	32	32	34	31	33	34	34	40	41	-	-	-
CASE TOTALS		EBG	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2.00	VERSED	mg	2.00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10.00	FENTANYL	cc	1.00	-	9.00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
125	DEMTOTHAL	mg	-	-	125	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10.00	EPHEDRINE	mg	-	-	-	-	10.00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10.00	NORCURON	mg	-	-	6.00	-	-	-	-	-	2.00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1000	CEFAZOLIN	mg	-	-	-	1000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5.00	ENLON DLUS	cc	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5.00	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Infusions		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	NITROGLYC u/K/M		-	1	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--	--»--
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Input-Output		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	LACTATED RINGRS		800	-	-	-	-	1000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
275	Urine Output		-	-	-	-	-	-	-	150	25	-	-	-	-	-	-	-	-	-	-	-	-	100	-	-
200	Est Blood Loss		-	-	-	-	-	-	-	-	150	-	-	-	-	-	-	-	-	-	-	-	-	200	-	-
	Other Losses		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CASE DATA	TEMP C. DEGREES		-	36.9	-	36.7	36.7	36.9	36.7	36.7	36.4	36.4	36.2	36.1	35.9	35.6	35.5	35.4	35.5	35.4	35.3	35.2	35.3	-	-	-
	TIDAL VOLUME	CC	-	-	-	1200	805	1106	893	790	800	805	815	810	800	804	803	812	804	800	805	250	340	-	-	-
	PIP	CM H20	-	-	-	25	32	21	23	19	22	23	22	21	24	24	23	22	24	22	23	-	-	-	-	-
	CVP	MM HG	-	6	-	3	8	7	7	2	-	6	-	8	-	-	10	8	8	10	8	5	6	-	-	-
	PA SYS BP	MM HG	-	21	-	20	27	31	29	29	26	26	27	25	24	22	23	24	26	28	24	25	24	-	-	-
	PA DIA BP	MM HG	-	10	-	10	16	16	16	15	14	14	15	12	12	10	11	11	18	17	14	13	14	-	-	-
	PA WEDGE	MM HG	-	4	-	-	16	-	-	12	-	-	14	-	-	-	9	-	-	15	-	-	12	-	-	-
	CARD OUTPUT	LPM	-	3.5	-	-	4.1	-	-	3.3	-	-	-	4.5	-	-	4.1	-	-	4.6	-	-	4.8	-	-	-
	Mixed Venous o2		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	BP-SOURCE	CODE	-	A	A	A	A	A	N	A	A	A	A	A	A	A	A	A	A	A	A	A	A	-	-	-
	HR-SOURCE	CODE	-	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	-	-	-
	ECG	RHYTHM	-	SR	SR	SR	SR	SR	SR	SR	SR	SR	SR	SR	SR	PVCs	SR	SR	PVCs	SR	SR	SR	SR	SR	SR	-

COMMENTS:

18:33- ARRIVED OR, ID CONFIRMED, RE-EVALUATED, MONITORS APPLIED
18:34- ARRIVED FROM ICU WITH ART LINE AND SWAN IN PLACE
18:50- ANTIBIOTIC PER SURGEON REQUEST
19:02- NASAL GASTRIC TUBE INSERTED PER REQUEST
19:04- FOLEY CATHETER INSERTED
19:55- CLOSING

Printed on 07/17/2005

Figure D.2. Sample Anesthesia Record - Page 2

REFERENCES

1. Jensen, C. and L. Feng and R.T. Snodgrass, *"Temporal Data Management"*. IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 1, January/February 1999.

2. Combi, C. and Y. Shahar, *"Temporal Reasoning and Temporal Data Maintenance in Medicine: Issues and Challenges"*. Computers in Biology and Medicine, Vol. 55, No. 5, pp 353-68, September 1997.

3. Goralwalla, I. A. and M.T. Özsu and D. Szafron, *"An Object-Oriented Framework for Temporal Data Models"*. Temporal Databases, Dagstuhl, pp 1-35, 1997.

4. Abello, A. and C. Martin, *"A Bitemporal Storage Structure for a Corporate DataWarehouse"*. International Conference on Enterprise Information Systems, April 2003.

5. Jensen, C.S. PhD Thesis: *"Temporal Database Management"*. http://www.cs.auc.dk/~csj/Thesis/, April 2000.

6. Health Level Seven, http://www.hl7.org/.

7. Gardner, M. and T. Peachey, "*A standard XML Schema for computerized anaesthetic records"*. Anaesthesia, Vol. 57, Issue: 12, Page 1174, December 2002.

8. Feldman, J.M. "*Medico-legal Aspects of Anesthesia Information Management Systems"*. Seminars in Anesthesia, Peri-operative Medicine and Pain Vol. 23, No: 2, Page 86-92, June 2004.
9. Jensen, S.J. and C.E. Dyreson, and "*The Consensus Glossary of Temporal Database Concepts-February 1998 Version"*. Temporal Databases, Dagstuhl 1997, Page 367-405, 1997.

10. Shahar, Y. "*Timing is Everything: Temporal Reasoning and Temporal Data Maintenance in Medicine"*. AIMDM 1999, Page 30-46, 1999.

11. Knolmayer, G.F. and T. Myrach, "*Concepts of Bitemporal Database Theory and Evolution of Web Documents"*. HICSS 2001.

12. Stantic, B. and G. Governatori, and A. Sattar, "*Handling of Current Time in Native XML Databases"*. ADC 2006, Vol. 49, Page 175-182, 2006.

13. Pornphol, P. and S. Chittayasothorn, "*A Temporal Relational and Object Relational Database Design Technique"*. SoutheastCon 2004, Page 54-59, 2004.

14. Kim, S.K. and S. Chakravarthy, "*Modelling Time: Adequacy of Three Distinct Time Concepts for Temporal Databases"*. International Conference of the Entity-Relationship Approach 1993, Page 475-491, 1993.

15. Combi, C. and A. Montanari, "*Data Models with Multiple Temporal Dimensions: Completing the Picture"*. CAiSE 2001, Page 187-202, 2001.

16. Bray, T. and J. Paoli and C.M. Sperberg, and E. Maler and F. Yergeau and J. Cowan, "*W3C: Extensible Markup Language (XML) 1.1."*. W3C Recommendation 16 August 2006, edited in place 29 September 2006, Available: http://www.w3.org/TR/xml11/, 2006.

17. Fallside, D.C. and P. Walmsley, "*XML Schema Part 0: Primer Second Edition"*. W3C Recommendation 28 October 2004, Available: http://www.w3.org/TR/xmlschema-0/, 2004.
18. Thompson, H.S. and D. Beech and M. Maloney and N. Mendelsohn, "*XML Schema Part 1: Structures Second Edition"*. W3C Recommendation 28 October 2004, Available: http://www.w3.org/TR/xmlschema-1/, 2004.

19. Biron, P.V. and K. Permanente and A. Malhotra, "*XML Schema Part 2: Datatypes Second Edition"*. W3C Recommendation 28 October 2004, Available: http://www.w3.org/TR/xmlschema-2/, 2004.

20. Biron, P.V. and K. Permanente and A. Malhotra, "*Document Object Model (DOM)"*. W3C Recommendation 28 October 2004, Available: http://www.w3.org/DOM/, 2004.

21. Wood, L. and A. L. Hors and V. Apparao and S. Byrne and M. Champion and S. Isaacs and I. Jacobs and G. Nicol and J. Robie and R. Sutor and C. Wilson, "Document Object Model (DOM) Level 1 Specification (Second Edition)" W3C Recommendation ,*1 October, 1998.*

22. Available: http://www.saxproject.org/.

23. Berglund, A. and S. Boag and D. Chamberlin and M.C. Fernández, and M. Kay and C. Robie and J. Siméon, "XML Path Language (XPath) 2.0" W3C Proposed Recommendation ,*21 November, 2006.*

24. Zhang, S. and C.E. Dyreson, "*Adding Valid Time to XPath"*. DNIS 2002, Page 29-42, 2002.

25. TimeCenter, http://www.cs.aau.dk/TimeCenter/index.htm

26. TimeDB, http://www.timeconsult.com/TimeConsult.html

27. "*Oracle 9 I Flashback Query"*. Oracle White Paper March 2002, Available: http://www.oracle.com/technology/deploy/availability/pdf/FlashbackQueryBWP.pdf, 2002.

28. Weiss, R. "*How Oracle Database 10 G Revolutionizes Availability and Enables The Grid"*. Technical Paper, Available: http://www.oracle.com/technology/tech/grid/collateral/10gAvailability.pdf.

29. LogExplorer, http://www.lumigent.com/products/le_sql.html

30. eXist, http://exist.sourceforge.net.

31. Xindice, http://xml.apache.org/xindice/.

32. Sedna, http://modis.ispras.ru/sedna/index.htm.

33. Natix, http://www.dataexmachina.de/natix.html.

34. Timber, http://www.eecs.umich.edu/db/timber.

35. Wang, F. and C. Zaniolo, "*XBIT: An XML-based Bitemporal Data Model*". ER 2004, Page 810-824, 2004.

36. Amagasa, T. and M. Yoshikawa and S. Uemura, "*A Data Model for Temporal XML Documents*". DEXA 2000, Page 334-344, 2000.

37. Grandi, F. and F. Mandreoli, and P. Tiberio, "*Temporal modeling and management of normative documents in XML format*". Data and Knowledge Engineering, Vol. 54, Issue. 3, September 2005, Page: 327-354, 2005.

38. Buneman, P. and S. Khanna and K. Tajima and W. Tan, "*Archiving scientific data*". ACM Sigmod Int. Conference, 2002, Page: 1-12, 2002.

39. Marian, A. and S. Abiteboul and G. Cobena and L. Mignet, "*Change-Centric Management of Versions in an XML Warehouse*". VLDB Conference, 2001, Page: 581-590, 2001.

40. Vaisman, A. and A.O. Mendelzon and E. Molinari and P. Tome, "*Temporal XML: Data Model, query language and implementation*". Available: http://www.cs.toronto.edu/~avaisman/papers.html, 2004.

41. Combi, C. and A. Montanari, "*Querying Data with Multiple Temporal Dimensions*". CAISE 2002, Page: 711-714, 2002.

42. Zhang, D. and V.J. Tsotras and B. Seeger, "*Efficient Temporal Join Processing Using Indices"*. ICDE 2002, Page 103-, 2002.

43. Parmar, V. "*Role of Information System and Automated Record in Anaesthesia"*. Indian Journal of Anaesthesia, Vol. 50, Issue: 2, pp 99-102, February 2006.

44. Salzberg, B. and V.J. Tsotras, "A *Comparison of Access Methods for Time-Evolving Data*". ACM Computing Surveys, Vol. 31, No: 2, June 1999.

45. Bliujute, R. and C. Jensen and S. Saltenis and G. Slivinskas, "R-*Tree Based Indexing of Now-Relative Bitemporal Data*". VLDB 1998, Page 345-356, 1998.

46. Guttman, A. "R-*Trees A Dynamic Index Structure for Spatial Searching*". Proc. ACM SIGMOD Intl. Conf. on Management of Data 1984, Page 47-57, 1984.

47. Beckmann, N. "The R*-*Trees An Efficient and Robust Access Method for Points and Rectangles*". Proc. ACM SIGMOD Intl. Conf. 1990, Page 322-331, 1990.

48. Cui, Y. "*High Dimensional Indexing*". Lecture Notes in Computer Science (LNCS). Vol. 2341 (Monograph) 2003.

49. Feng, Y. and A. Makinouchi, "*Efficient Evaluation of Partially-Dimensional Range Queries Using Adaptive R*-tree*". DEXA 2006, Page 687-696, 2006.

50. Kumar, A. and V.J. Tsotras and C. Faloutsos, "*Designing Access Methods for Bitemporal Databases*". IEEE Transactions on Knowledge and Data Engineering, Vol. 10, No.1, January/February 1998.

51. Khin-Myo, W. and Ng. Wee-Keong and L. Ee-Peng, "*ENAXS: Efficient Native XML Storage System*". APWeb 2003, Page 59-70, 2003.

52. Mendelzon, A.O. and F. Rizzolo and A. Vaisman, "*Indexing Temporal XML documents*". VLDB 2004, Page 216-227, 2004.

53. Duong, M. and Y. Zhang, "LSDX: A New Labelling Scheme for Dynamically Updating XML Data". Australian Database Conference 2005.

54. Li, Q. and B. Moon, "Indexing and Querying XML Data for Regular Path Expressions", VLDB 2001.

55. Cohen, E. and H. Kaplan and T. Milo, "Labelling dynamic XML trees", PODS 2002.

56. Dale, N. and D. Teague, "C++ Data Structures", Jones and Bartlett Computer Science, Second Edition.

57. A Computer Assisted Anaesthesia Record Keeper, http://www.rapid-record.com/index.htm