AN AUTOMATED CONVERSION OF TEMPORAL DATABASES INTO XML WITH FUZZINESS OPTION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF MIDDLE EAST TECHNICAL UNIVERSITY
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖMER ÖZGÜN IŞIKMAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER, 2010

Approval of the thesis:

AN AUTOMATED CONVERSION OF TEMPORAL DATABASES INTO XML WITH FUZZINESS OPTION

submitted by **ÖMER ÖZGÜN IŞIKMAN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Prof. Dr. Faruk Polat _____
Supervisor, **Computer Engineering Dept., METU**

Assist. Prof. Dr. Tansel Özyer _____
Co-supervisor, **Computer Engineering Dept., TOBB ETU**

Examining Committee Members:

Assoc.Prof.Dr. Ahmet Coşar
Computer Engineering Dept., METU _____

Prof. Dr. Faruk Polat
Computer Engineering Dept., METU _____

Assoc.Prof.Dr. Bülent Tavlı
Computer Engineering Dept., TOBB ETU _____

Assist. Prof. Dr. Tansel Özyer
Computer Engineering Dept., TOBB ETU _____

Dr. Çağatay Ündeğer
SimBT _____

Date: **03.09.2010**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ÖMER ÖZGÜN IŞIKMAN

Signature :

ABSTRACT

AN AUTOMATED CONVERSION OF TEMPORAL DATABASES INTO XML WITH FUZZINESS OPTION

Işıkman, Ömer Özgün
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. Faruk Polat
Co-Supervisor : Assist. Prof. Dr. Tansel Özyer

September, 2010, 54 pages

The importance of incorporating time in databases has been well realized by the community and time varying databases have been extensively studied by researchers. The main idea is to model up-to-date changes to data since it became available. Time information is mostly overlaid on the traditional databases, and extensional time dimension helps in inquiring for past data; this all becomes possible only once the idea is realized and favored by commercial database management systems. Unfortunately, one disadvantage of the temporal database management system is that it has not been commercialized. Firstly XML (eXtensible Markup Language) is a defacto standard for data interchange and hence integrating XML as the data model is decided. The motivation for the work described in this thesis is two-fold; transferring databases into XML with changing crisp values into fuzzy variables describing fuzzy sets and second bitemporal databases form one interesting type of temporal databases. Thus, purpose is to suggest a complete automated system that converts any bitemporal database to its fuzzy XML schema definition. However, the implemented temporal database operators are database content independent. Fuzzy elements are capable of having different membership functions and varying number of linguistic variables. A scheme for determining membership function parameters is proposed. Finally, fuzzy queries have also been implemented as part of the

system.

Keywords: fuzzy xml,bitemporal database, xml database, fuzzy queries, XML queries

ÖZ

BULANIKLILIK ÖZELLİKLİ ZAMANSAL VERİ TABANLARINDAN XML'E OTOMATİKLEŞTİRİLMİŞ ÇEVİRME

Işıkman, Ömer Özgün
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Faruk Polat
Ortak Tez Yöneticisi : Assist. Prof. Dr. Tansel Özyer

Eylül, 2010, 54 sayfa

Veritabanlarında zamanı birleştirmenin önemi bilgisayar çevrelerince iyi anlaşılmış, zaman değişkenli veritabanları araştırmacılar tarafından kapsamlı olarak araştırılmıştır. Genel düşünce veri uygun olana kadar güncel değişimleri saklayabilmektir. Zaman bilgisi genellikle geleneksel veritabanlarına bindirilmiştir ve eklenmiş zaman boyutu eski veriyi bulmakta yardımcı olabilirler; tüm bunlar düşünce günümüzde kullanılan veritabanı yönetim sistemleri tarafından incelenip kullanmaya başlandığında elveişli hale gelir. Ne yazık ki, zamansal veritabanı sisteminin bir dezavantajı günümüz veritabanlarında kullanılmıyor olmasıdır. Öncelikle veri değişiminde XML kabul edilmiş bir standart olduğu için XML kullanmaya karar verdik.Bu tezde tanımlanan işin amacı iki türlüdür; veritabanlarındaki sabit ve bulanık olmayan alanların XML'e bulanık mantık kurallarıyla aktarılması ve de zamana bağlı veritabanı türlerinden çift zamanlı veritabanı modelini kullanmaya karar verdik. Böylece, bizim amacımız, çift zamanlı veritabanını onun bulanık XML şema tanımına dönüştüren otomatikleştirilmiş bir sistem önermektir. Diğer taraftan, gerçekleştirilmiş zamana bağlı veritabanı operatörleri veritabanı içeriğinden bağımsızdır. Bulanık elementler farklı üyelik işlevlerine ve değişen dilsel değişken sayısına sahip olabilir. Bulanık mantık sisteminin kullanılması için her bir veritabanı için bir şablon oluşturulmasının iyi olacağına karar verdik. Son olarak, bulanık sorgular

sistemimizin bir parçası olarak ayrıca gerçekleştirildi.

Anahtar Kelimeler: bulanık XML, XML veritabanı,zamana bağlı veritabanı, XML sorguları

ACKNOWLEDGMENTS

I would like to show my gratitude to Prof. Dr Faruk Polat and my co-supervisor Assist. Prof. Dr. Tansel Özyer who made this thesis possible.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

1.1 Definition

Most of the current commercialized database management systems are widely implemented based on the conventional relational database theory with some limited extensions. Keeping only the most up-to-date snapshot of the data is one conspicuous weakness of commercial relational database management systems. In other words, they ignored the time aspect of the data, and this constraint the system to the most recent state [6]. Previous data is lost after the updating transaction commits. This drawback was realized by the database research community since late 80s and early 90s of the previous century. Several efforts contributed to the development of well established basis for temporal relational databases. Unfortunately, such initiatives have not been absorbed by the industry; accordingly the interest and investment in temporal databases declined in the academia. Temporal databases are important for many application domains. Their importance became more prominent due to the development of automated mining techniques that are capable of analyzing historical data for trend discovery among others. In other words, there are several applications that need the existence of actual temporal databases such as financial applications (portfolio, accounting, banking, record keeping, and banking), airline, train and hotel reservations, inventory and project management, scientific applications, weather monitoring, medical records [3], satellite, and multimedia data, among others. In some individual efforts, the time dimension is simulated indirectly in a way to partially satisfy these applications; however, a natural and native temporal database would be more satisfactory as comprehensive solution.

Because commercial databases are not capable of directly handling temporal data, tempo-

ral databases can be partially simulated and stored in relational databases. There are several attempts to extend SQL from temporal point of view and this requires time to be explicitly defined in the database. Several query languages specific to temporal database models are described in the literature, such as TOSQL [9], TQuel [10], HTQUEL [12], ETQL [13], and TSQL2 [11]. It is argued that temporal databases could have found their way to commercialization shall key end-users who are in need for database management systems with actual temporal capabilities know about the extensive and well established existing literature; they could have pushed for incorporating temporal aspects in commercial databases. Unfortunately, the gap between the industry and the academia remained unfilled.

Date and time have been introduced by SQL3 with some proposals for temporality. At the beginning, it was intended for object relational models particularly; and later on, SQL/Temporal has also been proposed in SQL3 [7, 8]. Other than temporal relational databases, researchers continued their efforts to extend new data models with temporal capabilities. For instance, research on temporal object-oriented models was active in the 90s, e.g., [43, 44, 45, 46, 47]. And recently, with the wide spread of the Extensible Markup Language (XML) [48], some research efforts focus on XML based temporal databases, e.g., [21, 22, 23, 20].

XML is the recommendation of W3C; it was derived from SGML. It is text based and simple for platform independent data interchange, mostly over the internet for recent web-based applications. In other words, with the advent of the internet technology, XML is being used for data interchange seamlessly without considering platforms. Realizing the uprising interest in XML and its influence, the work described in this thesis is the effort to develop a native temporal database management system by considering both relational and XML models. the object-oriented model is left outside the scope of the research because The current commercial relational database management systems have already absorbed all the attractive features of the object-oriented technology and turned mostly into object-relational databases even if it not explicitly declared. Also, current commercial database management systems are being extended with XML features. Therefore, considering both relational and XML models would lead to a comprehensive study that covers the state of the art in the database technology. Hence, temporal relational databases and XML is focused on. This model have been further extended and incorporated fuzziness in data. It is argued that a database system extended with fuzzy capabilities is another need that has been ignored by the database industry. The daily life is all based on fuzziness because humans by nature think and communicate in fuzzy

terms. Accordingly, naive users find it hard if at all possible to deal with crisp data; these naive users understand terms rather than values; telling someone that he/she scored 90 may not mean much to him/her; but telling the same person that he/she scored high will be more appreciated. Therefore, the flexibility of fuzziness gives lots of opportunities to embed real life characteristics in data. Possibilistic fuzzy distributions provide smooth mapping of fuzzy relational databases into fuzzy XML [35].

This thesis is mainly concentrated on one specific type of temporal databases, namely bitemporal databases which simply keep two time stamps for data, valid time and transaction time. A system is proposed that is capable of automate the transformation of relational bitemporal database into XML schema regardless of the underlying database management system; a relational database schema is converted into XML schema; and any temporal database operator can be performed on the transformed temporal XML data. Further, a new extension to temporal XML data with fuzziness has been put in use. Here, data can be analyzed and a partition scheme for determining fuzzy sets have been proposed. Any shape for the fuzzy function can be used for linguistic variables. The system is capable of performing fuzzy/crisp inquiries from tables/tuples and rank them. Different aspects of the developed system are presented in the sequel.

The rest of the paper is organized as follows. Related work is summarized in Section II. Section III gives an overview of the system with details on the algorithms needed for XML schema conversion, XML data content generation, temporal database operators with XML, fuzzy queries on XML data, and a partitioning scheme proposal for determining fuzzy linguistic variables. Section IV is conclusions. Practical issues regarding the program iterface, configuration and database have also been given in appendices A, B, and C.

1.2 Benefits of XML

XML is a meta-markup language for text documents. Data are included in XML documents as strings of text. The data are surrounded by text markup that describes the data. XML's basic unit of data and markup is called an element. The XML specification defines exact syntax this markup must follow: how elements are delimited by tags, what a tag looks like, what names are acceptable for elements, where attributes are placed, and so forth. Superficially,

the markup in an XML document looks a lot like the markup in an HTML document, but there are some crucial differences. XML is a complement of HTML, not a replacement.

The real power of XML comes from the fact that with XML, not only can you define your own set of tags, but the rules specified by those tags need not be limited to formatting rules. XML allows you to define all sorts of tags with all sorts of rules, such as tags representing business rules or tags representing data description or data relationships.

1.2.1 Advantages and Disadvantages of XML

- It is as easy as HTML.
- XML is fully compatible with applications like JAVA, and it can be combined with any application which is capable of processing XML irrespective of the platform it is being used on.
- XML is an extremely portable language to the extent that it can be used on large networks with multiple platforms like the internet, and it can be used on handhelds or palmtops or PDAs.
- XML is an extendable language, meaning that you can create your own tags, or use the tags which have already been created.
- It is a platform independent language.
- It can be deployed on any network if it is amicable for usage with the application in use.
- If the application can work along with XML, then XML can work on any platform and has no boundaries.
- It is also vendor independent and system independent. While data is being exchanged using XML, there will be no loss of data even between systems that use totally different formats. Even though XML has many advantages, it has some drawbacks too.

The disadvantages of XML are;

- XML syntax is redundant or large relative to binary representations of similar data.

- The redundancy may affect application efficiency through higher storage, transmission and processing costs.

- XML syntax is verbose relative to other alternative 'text-based' data transmission formats.

- No additional data type support: XML provides no specific notion of "integer", "string", "boolean", "date", and so on.

- The hierarchical model for representation is limited in comparison to the relational model or an object oriented graph.

- XML is commonly depicted as "self-documenting" but this depiction ignores critical ambiguities. [49]

1.3 Related Work

1.3.1 Temporal databases and XML

Temporal relational databases have been separated into grouped and ungrouped temporal databases [5]. Tuple time-stamping and attribute time-stamping are used. Tuple time-stamping resorts to tuple redundancy. Every time update occurs, a new tuple is inserted into the table/relation. A tuple time-stamping relation is conferred as 1NF (First Normal Form). In attribute time-stamping, an attribute can have attached time stamp. It can be named as N1NF (Non-first normal form).

One particular type, so-called bitemporal databases, captures both valid time and transaction time. In the course of time, keeping both transaction time and life time of data is prominent for having a more realistic environment. The work described in [14] proposes tuple stamping bitemporal databases. Each tuple is associated with five attribute values; Snodgrass et al. [10] use four time attributes and proposed the TQuel temporary query language. The work described in [15] binds valid and transaction time for attribute time-stamping [16].

As pointed out in [17, 18, 16], nested structure has been used for nesting at arbitrary level and time stamps have been associated with attributes. The nature of XML has led us to use the

nested structure. Further, the hierarchical structure of XML provides a natural environment for temporally grouped data models [19].

The work described in [24] added a tag <valid> for valid time description for each XML document; and the work described in [26] introduced XML structure. Here, leaf nodes have alternative values where changes in XML have been compiled in one XML file. The work described in [27, 28] proposed time related extensions on XPath for querying with valid and transactional time. Wang and Zaniolo [22, 21] proposed schemes on using XML for temporal databases. They transformed time-aspect English queries into XQuery and provided performance evaluations between a native XML database and DB2 [23].

1.3.2 Converting Relational Databases into XML

Considering the fact that XML is simple, portable, and extensible standard format and flexible means for interchanging data between different mediums, several studies have been held on converting relational database into XML. It attracts researchers because heavy part of the data still resides in relational databases. Several conversion algorithms are capable of converting from XML to relational database and vice a versa.

The main focus of interest is to extract the XML schema, rather than Document Type Definition (DTD), with the capabilities described in this thesis. DTD is left out because DTD uses extended context-free grammar which has limited data types and lacks the ability to add constraints. On the other hand, XML schema is enriched with more capabilities and different data types, type definitions, structural definitions. Also, XML schema definition is in XML notation. Applications such as those described in [31, 32, 33] have been proposed. Mainly, relational database to XML conversion is a systematic process that ends up with mapping tables and attributes in relational databases into elements and attributes by creating XML hierarchies and processes in application specific way [29, 30]. Here, while relational databases are flat, normalized and often proprietary; XML could be nested, un-normalized and public. The studies described in [29, 30] suggest an automated process of relation database to XML schema conversion; it is converted automatically regardless of learning tool specific languages.

1.3.3 Fuzzy XML

A classical set is a set with a crisp boundary, i.e., there is no unambiguous boundary. Membership degree of an element can take two values 0 or 1. In other words, an object is entirely in the set or not, whereas a fuzzy set as its name implies is a set without crisp boundaries. In other words, the transition from "belonging to a set" to "not belonging to a set" is gradual and this smooth transition is characterized by membership functions that give flexibility in modeling commonly used linguistic expressions. Fuzziness comes from the uncertain and imprecise nature of abstract thoughts and concepts [34].

Let X be the universe of discourse, if X is a collection of objects, each denoted generically by x then a fuzzy set A is defined as a set of ordered pairs:

$A = \{(x, \mu_A(x)) | x \in X\}$

where μ_A is called the membership function that maps each object x of domain X to a continuous membership value between 0 and 1, inclusive. There are two alternative ways to denote A:

$$A = \left\{ \begin{array}{ll} \sum_{x \in X} \mu_A(x)/x & \text{if} x \text{isdiscrete} \\ \int_x \mu_A(x)/x & \text{if} x \text{iscon tinuous} \end{array} \right\}$$

There are several classes of parameterized ways to define membership functions such as triangular, trapezoidal, Gaussian and bell functions.

Impreciseness in data necessitates incorporating fuzziness in XML. Storing crisp data in XML lacks the imprecise nature of the real world. It eventually lacks the ability to interface with fuzzy databases. There are two different approaches for storing data with imprecision: (1) Storing crisp data and doing imprecise queries, and (2) Storing imprecise data [23]. A third novel approach, developed at the University of Calgary by the research group of some coauthors of this thesis, introduced an intermediate fuzzy layer which absorbs all the fuzziness and takes the responsibility to map user requests into traditional queries [36]. The study described in [36] also suggested automated tuning of fuzzy membership functions for specific triangular shaped fuzzy sets. The second approach is implemented in the temporal database system described in this thesis; the fuzziness in XML is incorporated. There are considerable number of proposals dealing with this problem, e.g., [35, 37, 38, 39, 40, 41, 42]. These studies

```
<SALARY value="88958.0">
    <MEDIUM value="0.95"/>
    <HIGH value="0.05"/>
</SALARY>
```

Figure 1.1: Representation with single fuzzy tag

```
<SALARY fuzzy="on">
    <VALUE>88958.0</VALUE>
    <FUZZY name="medium">0.95</FUZZY>
    <FUZZY name="high">0.05</FUZZY>
</SALARY>
```

Figure 1.2: Representation with fuzzy set tag

follow two ways, they modify either the DTD or the XML Schema description.

1.3.3.1 Representation of Fuzzy XML

As it is seen in the literature, there is not a convention about representation about fuzziness in XML. There are some commonly used approaches in the research papers. They have some differences according to the topic they have mentioned about. It is good to show that with two different examples.

The representation seen on Figure 1.1 is commonly use for generic applications. The fuzzified column is set with a tag. Inner nodes are the fuzzy set matches of this object. This representation is used in this project. That is why it is going to be described in a detail way at next parts.

Figure 1.2 is commonly used for specific application. At these type of applications the limits of the related database is well-known. The fuzzy sets and linguistic variables are set before the database created. The schema cannot be changed softly. On the other hand, this is fast to be looked over as it is thought on XML query perspective. Because iteration on tags are more basic than iteration on attributes. The project does not contain well-bounded databases. Thus this type of usage is not chosen.

Although these representations are completely different from each other, the aims are same - store data. At this point, the confusing part becomes clear. If the aim is the same, there is only one reason to display in different ways. Easiness of use is the key. Every application has its own unique behavior. Everybody wants to use this easiness. However representing same thing with several ways is not a good strategy, especially in computer society.

CHAPTER 2

FUZZINESS IN XML

2.1 Fuzzy XML

Collecting data in XML is very usable because of advantages listed above. If we have a lot of information on our hand, the main problem is to handle it. Making simple searches is very important. Fuzziness becomes important at this point. If the data is represented in a fuzzy way, the flexibility will bring lots of improvements and different solutions. The main problem is to handle this fuzziness in XML at large amount of data.

Everybody cannot know syntax for every type of database. Because of that, using linguistic variables is the main task in fuzziness. At that point, the question ”how” needs to be answered.

Explaining XML in a theoretical way is not easy. Instead of that, describing the system and the structure on an example is a good approach. That is why, in this chapter, fuzziness in XML is going to be explained with an example XML database. The topics will be exampled on this database, after their theoretical parts are described.

2.1.1 Fuzzification of XML

It is important to represent fuzzy logic on XML. If the optimal solution is found to display fuzzy logic, it will be very easy to use fuzzy XML for application. The best solution for that is to keep in special fuzzy nodes. Three main reasons can be listed for this. Firstly, they are easy to construct. Secondly, it is easy to maintain and update these types of nodes. Finally, they are very simple and easy to understand the usage of it.

If an XML tuple is fuzzified, its XML schema changes according to the chosen fuzzified columns. In order to implement fuzzified elements, extra attributes and nodes are added to the node. With help of these changes the XML is easily represented and queried in a fuzzy way. Three main things change in a node are:

1. Fuzzy attribute: This attribute can be found at the XML node by default. Unless this exists, ”fuzzy” this attribute is added to relative node and the value of the attribute is set as ”on”. While an algorithm used for fuzziness is checking the node whether it is fuzzy or not, it checks that attribute. If its value is ”off” or this attribute does not exist, the algorithm behaves like a crisp value.

2. Fuzzy tag: This tag used to describe membership value of the current node. This node is set as child node of the original node. There can be more than one fuzzy tag in one node due to size of different membership values. It has an attribute called ”name” which represents the name of membership function. According to this value, the inner text displays the function value of the current membership function on the current node.

3. Value tag: Although this tag is not necessary for a fuzzified node, it is better to be used for precise queries. This is also added as a child node for the original node. If there is a value tag in a node, directly this node is used to be looked for a crisp search. Otherwise the value is found from fuzzified tags.

2.2 Membership Functions

The critical point on fuzzifying logic is to determine membership functions. Experts' opinions may give the most efficient solutions for functions. However finding an expert for each problem is not efficient and cost-effective. Instead, standardized linear functions are used to be used for membership functions **??**. The essential point in creating membership functions is to use a trapezoid. A trapezoid is a four-cornered shape that bottom and top edges are parallel to each other. Thus, each corner can represent meaningful information about fuzziness. Linear solutions are available with using 4 corners of that shape. Each of these corners has a match on fuzzy membership function. The 4 points are listed below.

1. Start Point: This is the left bottom corner of the trapezoid. The meaning of the point is

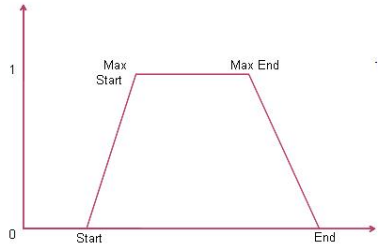

Figure 2.1: Trapezoidal Membership Function

the minimum value for the fuzzy set. Unless this point is ignored the functions value becomes more than zero after that point.

2. Maximum Start Point: This is the left top corner of the trapezoid. The meaning of the point is that value of the membership function becomes 1 starting from this point. This point also called ”minimum maximum point”. The function breaks after maximum start point.

3. Maximum End Point: This is the right top corner of the trapezoid. The meaning of the point is that this is the last point which membership function value becomes 1. The function breaks after maximum end point. After maximum end point, the membership value decrease toward to zero unless end point is ignored.

4. End Point: This is the right bottom corner of the trapezoid. This is the last point for the fuzzy set. Unless this point is ignored the functions value becomes zero after this point.

2.2.1 Types of Membership Functions

There are mainly 4 types of membership function are described in this thesis.

2.2.1.1 Trapezoidal

This is the basic function used in fuzziness in XML. All of the points used to describe a fuzzy set are defined separately. Each point is different from each other. As it is seen from Figure 2.1, maximum start and maximum end are different from each other and there is no ignored attribute available. The value of the function can be found from the similarity of triangles easily.

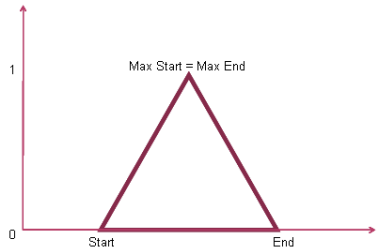

Figure 2.2: Triangular Membership Function

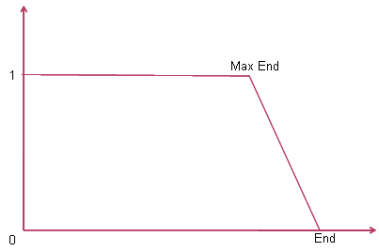

Figure 2.3: Minimum Starting Trapezoidal Membership Function

2.2.1.2 Triangular

This membership function is similar to trapezoidal. However, the maximum start point and maximum end point are same with each other. That is why there is only one maximum point. This is a very popular and commonly used technique in fuzzy logic. Generally, the maximum range of membership value is not too much. The similarity can be easily found with triangle similarity rules. The structure with a configuration example is shown in Figure 2.2.

2.2.1.3 Minimum Starting Point

This is similar to trapezoidal. On the other hand, there is an ignoring process for this type. If a value is less than maximum ending point it automatically returns as 1. This is very useful if you use it as first fuzzy set at ascending order. To set ignore attribute tag to ”start” is enough to construct this type of function. An example about structure and configuration is given in Figure 2.3.

2.2.1.4 Maximum Ending Point

This is similar to trapezoidal. On the other hand, there is an ignoring process for this type like minimum ending point. If a value is less than maximum starting point it automatically

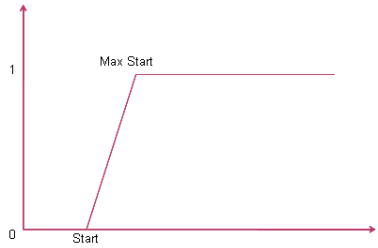

Figure 2.4: Maximum Ending Trapezoidal Membership Function

returns as 1. This is very useful if you use it as last first fuzzy set at ascending order. To set ignore attribute tag to ”end” is enough to construct this type of function. The structure with a configuration example is shown in Figure 2.4.

2.2.2 Configuration

Configuration file, named as ”config.xml” by default, is a file in which the fuzziness information kept. Configuring the fuzziness is the key point of the fuzzy logic system in XML. All of the fuzzy set and their membership information are kept in a single configuration file. This is a simple XML file in which these set and information are stored.

In a database system, there are schemas, tables and columns hierarchically. This file was constructed in this form. Firstly, the schema node is described. This node separates the databases from each other. After that the table is described inside of the schema. In a schema there may be lots of tables. Some of them may have same fuzzy sets. However, making the system complex is not a good idea. This separation provides that. Inside the table node, there are column nodes which are the real fuzziness part of the system. Similar to tables, there are separate fuzzy sets for each column. Every column has its own ”fuzzy_values” node. This node contains different fuzzy set for the relative column. If the membership functions are not set, fuzzy_values tag will be empty. The heart of the fuzziness is ”fuzzy” nodes. Every node keeps all of the necessary information to provide fuzziness about a fuzzy set with its attributes. The attributes and their features are listed below.

1. name attribute: This attribute keeps the name of the fuzzy set. This is the identifier for each fuzzy set for a column.

2. start attribute: This attribute keeps start point value for the fuzzy set.

3. maxStart attribute: This attribute keeps maximum start point value for the fuzzy set.

4. maxEnd attribute: This attribute keeps maximum end point value for the fuzzy set.

5. end attribute: This attribute keeps end point value for the fuzzy set.

6. ignore attribute: This attribute is not necessary for all the fuzzy sets. The main purpose of this attribute is to provide fuzziness correctness on starting and ending marginal values at the database. If a fuzzy tag has an attribute ignore, checks if it is start or end. If it is ”start” values of the membership functions up to maxEnd point become 1. If it is ”end” values of the membership functions from maxStart point becomes 1.

An example config.xml file is shown in B.

2.3 A Fuzzy Membership Proposal Scheme

Algorithm 1: PartitioningScheme(S,T,C,$FuzzySetTab$)

---

inputs: S % Schema name of the table

T % table name for the fuzzy column

C % Column to be fuzzified

$FuzzySetTab$ % An array with fuzzy set, types (linguistic variable and corresponding shape type)

{keeps the size of the parts divide total results}

$pCount \leftarrow 1$

{keep the values on the column}

$valList \leftarrow newList$

{number of elements in each piece}

$part \leftarrow 0$

{count to find correct piece}

$counter \leftarrow 0$

{a list keeps fuzzy set items}

$itemList \leftarrow newFuzzyItemlist$

```
for Row ∈ T do
    for V ∈ T do
        valList.add(value)
    end for
end for
sort(valList)
for (E ∈ FuzzySetTab) do
    if (E.type =′ Triangular′) then
        pCount ← pCount + 1
    else
        pCount = pCount + 2
    end if
end for
part = valList.size/Count
for (E ∈ FuzzySetTab) do
    {an object keep fuzzy set properties}
    item ← newFuzzyItem
    item.name ← E.fuzzyName
    if (E.type =′ MaxEndTrap′) then
        item.ignore ←′ end′
    else if (E.type =′ MaxStartTrap′) then
        item.ignore ←′ start′
    end if
    item.start ← valList.get(part ∗ (counter + +))
    item.maxStart ← valList.get(part ∗ (counter + +))
    if (E.type =′ Triangular′) then
        item.maxEnd ← item.maxStart
    else
        item.maxEnd ← valList.get(part ∗ (counter + +))
    end if
    item.end ← valList.get(part ∗ (counter + 1) − 1)
    if (item.start ≠ item.end) then
        itemList.add(item)
```

```
    end if
  end for
  return itemList
```

Columns that will be fuzzy are selected. According to Algorithm 3, partitioning is done as follows: A group of shapes (Figure 2.5) are picked for linguistic names and parameters. The number of linguistic variables and shapes are the same. Here, it is not claimed that, this proposed scheme promises near optimal fuzzy memberships and the memberships are not tuned. This scheme is subsidiary for having decision. Basically, there is the *pCount* variable. Shapes that are also sub-partitioned within themselves. Each partition is indexed from 1..*pCount* as an ordinal value.

The count of sub-partitions is calculated. Each shape has different number of partitions. Basically, there are four shapes, minimum start trap, max end trap, trapezoid, and triangle. All shapes, except triangle, have three partitions:

1. start-maxStart
2. maxStart-maxEnd
3. maxEnd-end

Minimum start trap has the same 'start' and 'maxStart' values (partition 1); and Max end trap has the same 'maxEnd' and 'end' values (partition 3). These are kept in three partitions because both ends of the shapes are still taken into account. Triangle doesn't have the maxStart-maxEnd values (partition 2). Because of this, triangle has only two partitions. While finding the pCount value, all shapes' number of partition values are summed and number of shapes minus one value is subtracted from that value. Because shapes except the last one overlap and according to thesis suggestion, only two shapes can overlap at once. For example, Figure 2.5 has 11 parts. In this scheme, there is the assumption that all linguistic variables are ordered according to their meanings (e.g., young<middle<old) and previous linguistic variable's *maxend* value is the same as the next linguistic variable's *start* value. At most only two linguistic variables can be greater than zero. All values of that fuzzy column are sorted in order. Each *pCount* has been considered as the bin of an equi-depth histogram so that

records are distributed equally and *start*, *maxstart*, *maxend*, *end* are determined accordingly. This scheme is a suggestion and subject to change unless used. Results of the partitioning is written in config.xml (Figure 2.6).

2.4 Fuzzy Querying in XML

Given the schema, table, restrictions in columns, columns to be projected, a fuzzy query is performed by invoking Algorithm 2. Each row of the table is controlled against the restrictions. If the score of the row is above the threshold then it is omitted. Otherwise, it is included in the list. If there is a crisp criteria in the query then the score of the row increases by one if there is a match. If it is fuzzy then, the $FuzzyVal$ (Algorithm 3) function is invoked for column C and restriction r. This function returns a decimal value from the interval [0,1]. The row has the score as the sum of the scores for each restriction. Rows are displayed in sorted order and columns to be projected are displayed for each row.

Algorithm 2: FuzzyQuery(S,T,$PCols$,R,Thr)

---

inputs: S % schema of the table for a query
T % Table as input of the query
$PCols$ % columns that will be displayed in the results
R % restrictions for the query
Thr % Threshold rank score to display in the list

$Results \leftarrow newResultList$
for ($Row \in T$) **do**
  $p \leftarrow 0$
  {An element keeps projected record}
  $Result \leftarrow newResultElement$
  for ($C \in Row$) **do**
    {Point for each column}
    $colp \leftarrow 0$
    {True if used for restriction}
    $used \leftarrow$ false

```
        for (r ∈ R) do
            if (C ∈ R.Column) then
                used ← true
                if (r.Typeisfuzzy) then
                    colp ← colpp + FuzzyVal(C, r)
                else if (C.Value = R.Value) then
                    colp ← colp + 1
                end if
            end if
            if (used = false) then
                colp ← colp + 1
            end if
        end for
        p = p + colp
        if ((PColsisnull) or (C ∈ PCols)) then
            Result.Add(C)
        end if
    end for
    Result.p ← p
    if (Result.p ≥ Thr) then
        Results.Add(Result)
    end if
end for
Results ← Sort(Results)
return Results
```

The *FuzzyValue* function maps the membership value according to the given linguistic variable and the other types of shapes that have been described in this system.

Fuzzy values are mapped with *fuzzyVal*. This function uses a config.xml file as in Figure 2.6. There is a detailed version of this configuration file in B It includes fuzzy membership settings for each fuzzy column. Figure 2.5 lists all types of shapes for describing fuzzy sets. They are namely, triangle, trapezoid, minimum start trap and maximum start trap. Each shape contains

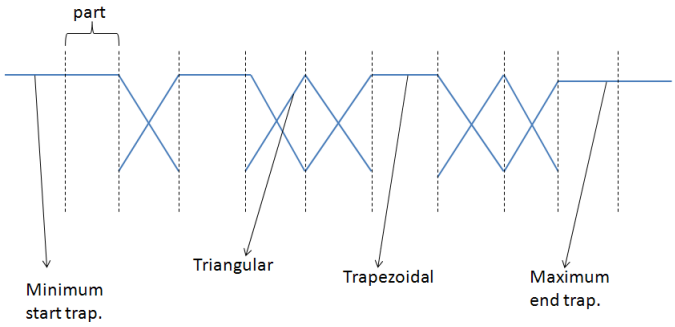

Figure 2.5: Fuzzy Shapes Defined in the System

```
- <CONFIG>
  - <SCHEMA name="EMPLOYEES">
    - <TABLE name="EMPLOYEES">
      - <COLUMN name="AGE">
        - <FUZZY_VALUES>
            <FUZZY end="50" ignore="start" maxEnd="45.0" maxStart="35.0" name="young" start="25.0"/>
            <FUZZY end="60" maxEnd="55" maxStart="50" name="middle" start="40.0"/>
            <FUZZY end="70.0" ignore="end" maxEnd="65.0" maxStart="60" name="old" start="55"/>
          </FUZZY_VALUES>
        </COLUMN>
      </TABLE>
    </SCHEMA>
  </CONFIG>
```

Figure 2.6: Fuzzy Membership Settings (config.xml)

start, *maxstart*, *maxend*, *end* values to represent the shape parameters; these are the x-axis values; *maxstart* and *maxend* give the range where the membership value is 1; *start* is the value where the shape starts from and *end* is the value where the shape ends; the membership value is zero for the latter two parameters. *start* and *end* values can be ignored for minimum start trap and maximum start trap shapes (see Figure 2.5).

Algorithm 3: FuzzyValue(C,r)

---

inputs: C % Column

r % A Restriction with name, value and type

$fuzzVal \leftarrow FindFuzzyRangeforrandC$

if ($fuzzValisnull$) **then**

  return 0

end if

if ($fuzzVal.ignore$ is *null*) or ($fuzzVal.ignore$ is not *start*) **then**

```
    if (fuzzVal.start > C.Val) then
        return 0
    else
        return (C.Val − fuzzVal.start)/
        (fuzzVal.maxstart − fuzzVal.start))
    end if
else
    if (C.Val ≤ fuzzVal.maxstart) then
        return 1
    end if
end if
if ((C.Val ≥ fuzzVal.maxStart) and (C.Val ≤ fuzzVal.maxend)) then
    return 1
end if
if ((fuzzVal.ignoreisnull) or (fuzzVal.ignoreisnotend )) then
    if (fuzzVal.end < C.Val) then
        return 0
    else
        return (fuzzVal.end − C.Val)/(fuzzVal.end − fuzzVal.maxend)
    end if
else
    if (C.Val ≥ fuzzVal.maxend) then
        return 1
    end if
end if
return 0
```

Fuzzy queries are coded using a special language. All queries start with show or display. Columns for display are delimited with “,” or “and”. The term “of” precedes the table name. Restrictions are optional and start with whose. Restrictions are split into sub restrictions. They can have conjunction (with “,”,“and”) or disjunction (with “or”). Restrictions can have the following relational operators:, equal, smaller than, bigger than, smaller than or equal,

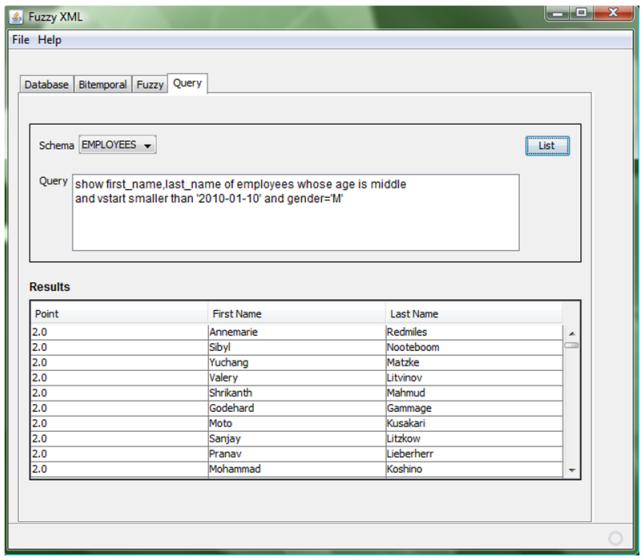

Figure 2.7: Fuzzy Query Screen

bigger than or equal, not, denoted =, <, >, <=, >=, ><, respectively.

Query = (“show”|“display”) [<column> (“,”|“and” <column>)* “of”] <table> [“whose” <Restriction List>]

<Restriction List> = <Restriction>[(“,”|“and”|“or”) <Restriction>]*

<Restriction> = column (“is” | “smaller than” | “bigger than” | “smaller than or equal” | “bigger than or equal”| “is not” | “between” | “=” | “<>” | ”>=” | “<=” |“><” | “<” | “>”) <columnOperand>

Example: show first_name, last_name of employees whose age is middle and VE of hire_date smaller than ‘2010-01-10’ and gender=‘M’

Here, first_name and last_name are columns to be listed; ‘employees’ is the name of the table; and restrictions are “age is middle”, “VE of hire_date smaller than ‘2010-01-10’”, and gender=‘M’. The column operand can be a fuzzy linguistic variable, bitemporal time attribute value.

Bitemporal comparison is used for filtering the result set. All restrictions except bitemporal time attribute comparisons can return the value 1 as maximum; 0 as minimum. In the example, the value can result in 2 as maximum (Figure 2.7) and the projected column of each record is displayed according to the score value in non-increasing order. According to compiled table, you can see the result set with varying scores in Figure 2.8.

point	first_name	last_name
2,00	Aiman	Constantine
2,00	Hyuncheol	Zolotykh
2,00	Angel	Mondadori
	...	
1,75	Renny	Pramanik
1,75	Masato	Setlzner
	...	
1,50	Shen	Brattka
1,50	Shay	Casley
1,50	Mang	Erie
	...	
1,25	Giap	Matzel
1,25	Duangkaew	Rassart
1,25	Fan	Fairtlough
1,25	Kirk	Kalsbeek
	...	

Figure 2.8: Query Results with Different Scores(config.xml)

CHAPTER 3

BITEMPORAL XML

3.1 Overview of TempoXML

A bitemporal database contains 4 time variables; these are: vStart, vEnd, tStart, and tEnd. This structure has been used to represent bitemporal time elements better and in a modular way. Initially, vEnd and tEnd have been taken as "now" while tStart and vStart have been taken as system specific initial minimum time value. Bitemporal time stamps are stored in special structure called *BitemporalTime*. Subsequently, this object type has been used for time-stamp instantiation of tuple or column. Each time-stamp object value is set with these four values. Here, the match with four values require them to be of date time type, and particularly selected for date and time interval bookkeeping. The instantiation of time-stamp information in an object of BitemporalTime class type is given in Algorithm4.

Algorithm 4: AssignBiTemporalElement(*node*)

Input: *node* % xml element

$bitemporalElement \leftarrow newBitemporalTime$

$bitemporalElement.vStart \leftarrow node.vStart$

$bitemporalElement.vEnd \leftarrow node.vEnd$

$bitemporalElement.tStart \leftarrow node.tStart$

$bitemporalElement.tEnd \leftarrow node.tEnd$

return $bitemporalElement$

TempoXML is the front-end tool that can connect to any database management system (DBMS);

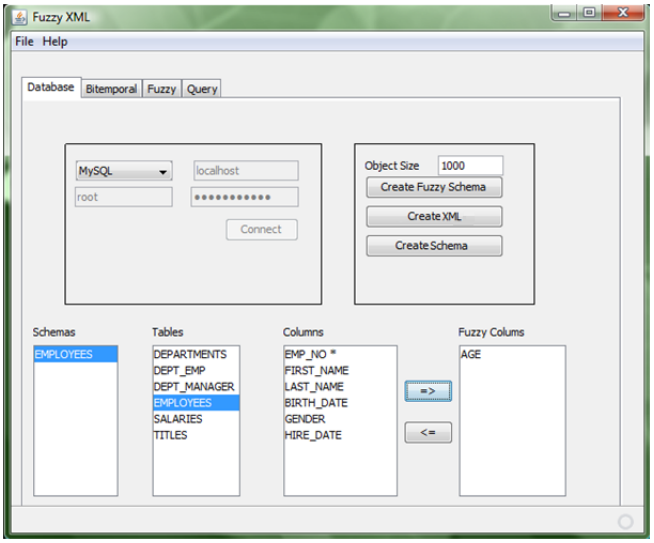

Figure 3.1: An Overview of the System

and hence allows us to use any existing database implemented using the underlying DBMS. A general view of the system is illustrated in Figure 3.1. This system can be used to create XML schema of the database tables. Tables, columns and fuzzy columns can be selected and the XML schema can be generated automatically. All operations identified in the application are database meta-data and column type aware. Based on the XML schema, the bitemporal database can be converted into XML document.

3.1.1 Conversion From BiTemporal Relational Data Model to XML Schema

Temporal relational database to XML schema conversion is fulfilled by employing the adapted methodology as applied in [29, 30, 35].

Algorithm 5: RelToSchema($Table$,C,$Fuzzy\_El$)

Input: $Table$ % table name of the selected schema

C % columns to be inserted into XML file

$Fuzzy\_El$ % columns to be transformed into fuzzy in XML

$schemaEl \leftarrow$ Create complex element for the schema

$tableEl \leftarrow$ Create complex Element for the Table Entity as TABLE

Insert tableEl into schemaEl sequence

Insert bitemporal attributes into tableEl

```
Insert id attribute into tableEl
Insert primary attribute into tableEl
for (C ∈ Table) do
  if (CisPrimaryTable) then
    primaryNode ← Create complex XML element for C
    Insert primary attribute into primaryNode
    Insert textContent for primaryNode
    Insert primaryNode into tableEl sequence
  else
    {CNode is Column Node and CContainer is Column Container}
    CContainer ← Create complex XML elements for C.name+"_VALUES"
    CNode ← Create complex XML element for C
    Insert bitemporal attributes into CNode
    Insert textContent for CNode
    Insert CNode into CContainer sequence
    Insert CContainer into tableEl sequence
  end if
end for
for (F ∈ Fuzzy_El) do
  fCContainer ← Create complex XML elements for F.name+"_VALUES"
  Insert fuzzy attribute into fCContainer
  fCNode ← Create complex XML element for F
  Insert bitemporal attributes into fCNode
  {Membership value assignment in schema definition starts here}
  fuzzyVal ← Create simple XML element for value of F
  Insert fuzzyVal into fuzzyColonNode
  fuzzySetEl ← Create complex XML element for fuzzy set elements
  {describe the name of the fuzzy set}
  Insert name attribute into fuzzySetEl
  Insert fuzzySetEl into fCNode
  {Membership value assignment in schema definition ends here}
  Insert textContent for fCNode
  Insert fCNode into fCContainer sequence
```

Insert fCContainer into tableEl sequence

end for

XML data is validated with the XML Schema standardized by W3C. The schema can be generated using *RelToSchema*. Table, columns with crisp values, and columns that are determined to be fuzzy are the input. XML has a nested structure and at the top of the structure, the name of the table is included as the root element. Inside that element, the table itself is inserted with "TABLE_"+ <*table name*> node. At this level, the four bitemporal time-stamp attributes are nested inside each element. Each row has an attribute for unique id, primary columns. After the primary column, non-fuzzy columns and fuzzy columns are defined in the schema. Columns definition is done with <*Column*>+"_VALUES" node. It has children nodes only. Each element inside the node has the bitemporal time stamp attribute. Also content value exists. Fuzzy columns have <*Column*>+"_VALUES" container node, and it differs with the "fuzzy" attribute only. Also this column has the fuzzy membership mapping. It contains two XML elements inside. One is its value given with the "VALUE" element. The other one is the "FUZZY" element(s) having name (linguistic variable) and membership value.

3.1.2 Conversion From BiTemporal Relational Data Model to XML Data

XML data can be created by the two procedures *RelToCrispXML* and *CrispXMLToFuzzyXML* given in Algorithm 6 and 7, respectively. At the *RelToCrispXML* stage, the inputs are the schema S, Table, Columns of Table to exist in $XML(C)$, columns that are identified as fuzzy(Fuzzy_El) and T representing four time-related columns used for time-stamping in the bitemporal database. After schema creation, the schema definition node is created, and it is referenced from inside the XML. Later, for each column, one node is created.

Bitemporal time-stamps ($vStart$,$vEnd$,$tStart$,$tEnd$) are used as attributes for table rows and table columns (fuzzy/nonfuzzy). The elements in the XML database can be generated according to the rules produced by Algorithm 6. An id attribute is given to each tuple of the table existing in XML. The related row is determined by using the id of the XML node. Basically, this id can be generated with the primary keys of the table. In case the number of keys is more than one then a composite primary string key is located with corresponding composite values with special delimiters. That node does not involve any time stamp information because it is

```
− <xs:schema>
  − <xs:element name="EMPLOYEES">
    − <xs:complexType>
      − <xs:sequence>
        − <xs:element name="TABLE_EMPLOYEE">
          − <xs:sequence>
              <xs:attribute name="vstart" type="xs:date"/>
              <xs:attribute name="vend" type="xs:date"/>
              <xs:attribute name="tstart" type="xs:date"/>
              <xs:attribute name="tend" type="xs:date"/>
              <xs:attribute id="xs:string"/>
              <xs:attribute primary="EMP_NO"/>
            − <xs:element name="EMP_NO">
              − <xs:complexType>
                − <xs:simpleContent>
                  − <xs:extension base="xs:integer">
                      <xs:attribute name="primary" type="xs:string"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            + <xs:element name="FIRST_NAME_VALUES"></xs:element>
            + <xs:element name="LAST_NAME_VALUES"></xs:element>
            + <xs:element name="BIRTH_DATE_VALUES"></xs:element>
            + <xs:element name="GENDER_VALUES"></xs:element>
            − <xs:element name="HIRE_DATE_VALUES">
              − <xs:complexType>
                − <xs:sequence>
                  − <xs:element name="HIRE_DATE">
                    − <xs:complexType>
                      − <xs:simpleContent>
                        − <xs:extension base="xs:date">
                            <xs:attribute name="vstart" type="xs:date"/>
                            <xs:attribute name="vend" type="xs:date"/>
                            <xs:attribute name="tstart" type="xs:date"/>
                            <xs:attribute name="tend" type="xs:date"/>
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            − <xs:element name="AGE_VALUES">
              − <xs:complexType>
                  <xs:attribute name="fuzzy" type="xs:string"/>
                − <xs:sequence>
                  − <xs:element name="AGE">
                    − <xs:complexType>
                        <xs:attribute name="vstart" type="xs:date"/>
                        <xs:attribute name="vend" type="xs:date"/>
                        <xs:attribute name="tstart" type="xs:date"/>
                        <xs:attribute name="tend" type="xs:date"/>
                        <xs:element name="VALUE" type="xs:string"/>
                      − <xs:sequence>
                        − <xs:element name="FUZZY">
                          − <xs:complexType>
                            − <xs:simpleContent>
                              − <xs:extension base="xs:string">
                                  <xs:attribute name="name" type="xs:string"/>
                                </xs:extension>
                              </xs:simpleContent>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Figure 3.2: Conversion of EMPLOYEE Relation Into XML Schema

assumed that the row has an identifier with its primary keys and those keys can not change in the course of existence. Each row keeps the bounds of the time-stamps of the comprising columns. If the column is not a bitemporal time ($C \notin T$) then for each node temporal changes are located in the XML element <C>._VALUES. Each temporal change is nested with T attributes. In case the column is fuzzy then an extra $fuzzy$ = "on" attribute indicates that fuzziness is added for the fuzzy column.

Algorithm 6: RelToCrispXML(S,$Table$,C,$Fuzzy\_El$,T)

Input:

S % name of the schema of the database

$Table$ % name of the table of the selected schema

C % columns to be inserted into the XML file

$Fuzzy\_El$ % columns to become fuzzy in XML

T % elements to be used for time-stamping

schemaEl ← Create Schema Node

tableEl ← Create Element or Table Entity as TABLE_<*Table Name*>

for $C \in Table$ **do**

  if I(C is T) **then**

    {check if the current column is a bitemporal element}

    if *C.type* is "vstart" **then**

      vstart = *C*

    else if *C.type* is "vend" **then**

      vend = *C*

    else if *C.type* is "vstart" **then**

      tstart = *C*

    else if *C.type* is "tend" **then**

      tend = *C*

    end if

  else if I(C is $Fuzzy$) **then**

    {check if the current column will be fuzzy in XML}

    columnEl ← Create Element for *C*

Insert Attributes *vstart*, *vend*, *tstart*, *tend* into *columnEl*

Fuzzy_El ← Create Element for "fuzz-values"

fuzzy ← Create Element Node for "fuzzy"

Insert Attributes "name" to Fuzzy

Set Type Member to Fuzzy

Insert *fuzzy* into *fuzzyEl*

Insert *fuzzyEl* into *columnEl*

valueEl ← Create Element Node for "value"

Insert *valueEl* into *columnEl*

Insert Attribute "fuzzy" into *columnEl*

Insert *columnEl* into *tableEl*

else

columnEl ← Create Element Node for *C*

Insert attributes *vstart*, *vend*, *tstart*, *tend* for *columnEl*

end if

end for

Insert Attributes *vstart*, *vend*, *tstart*, *tend* into *tableEl*

Insert *tableEl* into *schemaEl*

Example for a Employee is given in (Table 3.1).

Table 3.1: Employee

#EMP_NO	BIRTH_DATE	FIRST_NAME	LAST_NAME	GENDER	HIRE_DATE	AGE
10192	1960-09-16	Johua	Falck	M	1988-06-13	50

According to the *RelToCrispXML* algorithm and by using the Employee Relation, (Table 3.1), fuzzy column values are assigned by invoking *CrispXMLToFuzzyXML*. Here, *AGE* is required to be a fuzzy column; it is obtained that the XML data below with the bitemporal time attributes as shown in Figure 3.3.

Algorithm 7: RelToFuzzyXML(*Node*, *FS Map*)

```xml
<?xml version="1.0" encoding="UTF-8" ?>
- <TABLES>
  - <EMPLOYEES>
    - <TABLE_EMPLOYEE>
        <EMP_NO primary="true">11000</EMP_NO>
      - <BIRTH_DATE_VALUES>
          <BIRTH_DATE tend="now" tstart="now" vend="now" vstart="now">12-09-1960</BIRTH_DATE>
        </BIRTH_DATE_VALUES>
      - <FIRST_NAME_VALUES>
          <FIRST_NAME tend="now" tstart="now" vend="now" vstart="now">Alain</FIRST_NAME>
        </FIRST_NAME_VALUES>
      - <LAST_NAME_VALUES>
          <LAST_NAME tend="now" tstart="now" vend="now" vstart="now">Bonifati</LAST_NAME>
        </LAST_NAME_VALUES>
      - <GENDER_VALUES>
          <GENDER tend="now" tstart="now" vend="now" vstart="now">M</GENDER>
        </GENDER_VALUES>
      - <HIRE_DATE_VALUES>
          <HIRE_DATE tend="now" tstart="now" vend="now" vstart="now">20-08-1988</HIRE_DATE>
        </HIRE_DATE_VALUES>
      - <AGE_VALUES fuzzy="on">
        - <AGE fuzzy="on" tend="now" tstart="now" vend="now" vstart="now">
            <VALUE>50.0</VALUE>
            <FUZZY name="young">1.0</FUZZY>
            <FUZZY name="middle">0.75</FUZZY>
          </AGE>
        </AGE_VALUES>
      </TABLE_EMPLOYEE>
    - <TABLE_EMPLOYEE>
    </EMPLOYEES>
  </TABLES>
```


Figure 3.3: Conversion of the EMPLOYEE Relation into XML

Input: *Node* % table XML node at Crisp XML

FS Map % fuzzy sets for each column

tableEl ← clone(*Node*)

for (*Row* ∈ *tableEl*) **do**

  for (*N* ∈ *Row*) **do**

    if ((N.attribute('fuzzy')!='on') **then**

      Do nothing

    else

      colSet ← Get fuzzy set for *N* from FSMap

      for (*C* ∈ *N*) **do**

        C ← *null*

        {new Node for *C*.Val named "VALUE"}

        valNode ← new Node

        insertvalNodetoC

        for (*R* ∈ *colSet*) **do**

          $CVal \leftarrow fuzzyVal(C, R)$

          if ($CVal > 0$) **then**

$fNode \leftarrow$ Create "FUZZY" node for the current fuzzy set

$fNode.attr$["$name$"] $\leftarrow R.name$

$fNode.value = CVal$

$insertfNodetoC$

end if

end for

end for

end if

end for

end for

After schema generation, the XML data file is created. Accordingly, the temporal database operators on the XML data generated so far will be described. For the subsequent operators, a node search operation is performed for node access. According to the function, a set of rows with id values can be found; if the id is correct, this element is chosen as correct node. Eventually, this function returns a list of nodes satisfying id as an element of the given IDs. For example, findNodes('EMPLOYEEES', 'EMPLOYEEES',10192, 10191) returns employees $Emp\_No = 10191 \vee 10192$.

Algorithm 8: FindNodes($Schema$,$Table$,IDs)

---

inputs: Schema % refers to the database schema to look for Table % name of the table in the schema IDs % ids of the table elements, there can be unlimited size of elements

$nodes \leftarrow new$ array of size specified as the length of IDs {new array[$IDs.length$]}

$schemaNode \leftarrow$ Read schemas and find the required schema

$tableNode \leftarrow$ Read tables from schemaNode and find the required table

$counter \leftarrow 0$

for $tableRow \in tableNode$ **do**

for each $id \in IDs$ **do**

if $id = tableRow.attribute$("id") **then**

$nodes[counter] \leftarrow tableRow$

$counter \leftarrow counter + 1$

break

```
        end if
      end for
    end for
    return nodes
```

3.2 Temporal Database Operators in XML

Figure 3.4 summarizes union, difference, and intersection as temporal database operators according to four elements of bitemporal time stamp.

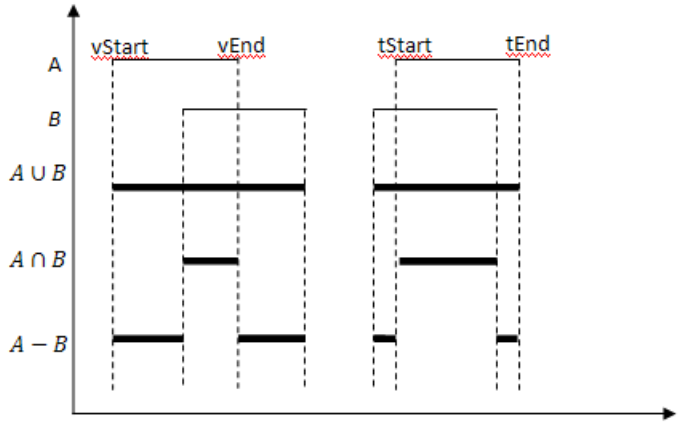

Figure 3.4: A Summary of Union, Difference, and Intersection

3.2.1 Union

This operation mainly finds the largest time interval between two bitemporal time elements (Figure 3.4. The first thing is to find the correct nodes by using *FindNodes*. In the algorithm, two ids of table rows, table, and schema must be given. The column parameter is optional. If it is null, the time-stamp comparison between rows is done, otherwise the column is considered only. In other words, this element can be found either in the entire node with sub-nodes that are uniquely identified with id value or in the specified column node. First of all, ids are given; and based on ids, related nodes are found with bitemporal time stamps. These time stamps are instantiated by invoking Algorithm 4. The union operator compares four elements of *BiTemporal* and after comparison, older $vStart$ and $tStart$ values and recent $vEnd$ and $tEnd$

values are returned. When a column is chosen, the union of this column (Algorithm 10) is found. Then, the union algorithm is applied on the found bitemporal elements. For Union, Algorithm 9 has been used to find the time interval.

Algorithm 9: Union($Schema$,$Table$,$Column$, $ID1$, $ID2$)

---

inputs: $Schema$ % refers to the database schema to look for
$Table$ % name of the table in the schema
$Column$ % name of the column in the table -not necessary
$ID1$ % id of the first table element
$ID2$ % id of the second table element

$nodes \leftarrow$ FindNodes($Schema$,$Table$,$ID1$,$ID2$)
{The first and second elements of nodes array}
$node1 \leftarrow nodes[0]$
$node2 \leftarrow nodes[1]$
$btTime \leftarrow newBitemporalTime$
if ($node1 = null$ and $node2 = null$) **then**
  return null
else
  if ($node2 = null$) **then**
    $node2 \leftarrow node1$
  end if
  if ($node1 = null$) **then**
    $node1 \leftarrow node2$
  end if
end if
$btTime1 \leftarrow AssignBiTemporalElement(node1)$
$btTime2 \leftarrow AssignBiTemporalElement(node2)$
if $Column \neq null$ **then**
  $btTime1 \leftarrow ColumnUnion(node1, Column)$
  $btTime2 \leftarrow columnUnion(node2, Column)$
end if

if $btTime1.vStart < btTime2.vStart$ **then**

$btTime.vStart \leftarrow btTime1.vStart$

else

$btTime.vStart \leftarrow btTime2.vStart$

end if

if $btTime1.tStart < btTime2.tStart$ **then**

$btTime.tStart \leftarrow btTime1.tStart$

else

$btTime.tStart \leftarrow btTime2.tStart$

end if

if $btTime1.vEnd < btTime2.vEnd$ **then**

$btTime.vEnd \leftarrow btTime2.vEnd$

else

$btTime.vEnd \leftarrow btTime1.vEnd$

end if

if $btTime1.tEnd < btTime2.tEnd$ **then**

$btTime.tEnd \leftarrow btTime2.tEnd$

else

$btTime.vEnd \leftarrow btTime1.vEnd$

end if

return $btTime$

The *ColumnUnion* function finds the largest time interval. All temporal changes of the column undergo a union operation. Inside the <Column>._VALUES.all bitemporal time stamps are iterated through to capture the oldest *vStart* and *tStart* values and the most recent *vEnd* and *tEnd* values.


```
- <SALARY_VALUES fuzzy="on">
    <SALARY fuzzy="on" tend="14-06-1993" tstart="14-09-1992" vend="14-06-1993" vstart="14-06-1992">67932</SALARY>
    <SALARY fuzzy="on" tend="14-06-1994" tstart="14-06-1993" vend="14-06-1994" vstart="14-06-1993">68046</SALARY>
    <SALARY fuzzy="on" tend="14-06-1995" tstart="14-06-1994" vend="14-06-1995" vstart="14-06-1994">67675</SALARY>
    <SALARY fuzzy="on" tend="13-06-1996" tstart="14-06-1995" vend="13-06-1996" vstart="14-06-1995">69876</SALARY>
    <SALARY fuzzy="on" tend="13-06-1997" tstart="13-06-1996" vend="13-06-1997" vstart="13-06-1996">70464</SALARY>
    <SALARY fuzzy="on" tend="13-06-1998" tstart="13-06-1997" vend="13-06-1998" vstart="13-06-1997">71574</SALARY>
    <SALARY fuzzy="on" tend="13-06-1999" tstart="13-06-1998" vend="13-06-1999" vstart="13-06-1998">74638</SALARY>
    <SALARY fuzzy="on" tend="12-06-2000" tstart="13-06-1999" vend="12-06-2000" vstart="13-06-1999">79102</SALARY>
    <SALARY fuzzy="on" tend="12-06-2001" tstart="12-06-2000" vend="12-06-2001" vstart="12-06-2000">82731</SALARY>
    <SALARY fuzzy="on" tend="12-06-2002" tstart="12-06-2001" vend="12-06-2002" vstart="12-06-2001">85931</SALARY>
    <SALARY fuzzy="on" tend="now" tstart="12-06-2002" vend="now" vstart="12-06-2002">86641</SALARY>
  </SALARY_VALUES>
```


Figure 3.5: Column Union

Figure 3.5 gives the temporal changes of the column Salary which is fuzzy. According to the given values, the column union returns [14-06-1992,now], [14-09-1992,now] for [vStart,vEnd] and [tStart,tEnd] time-stamp pairs.

Algorithm 10: ColumnUnion(*Column*, *Node*)

---

inputs: *Column* % tag of column attribute
Node % Table Entity of a node

$btTime \leftarrow newBitemporalTime$
{finds the children nodes which are named same with the column}
for $columnNodeinnode$ **do**
  if $columnNode.vStart < btTime.vStart$ **then**
    $btTime.vStart \leftarrow columnNode.vStart$
  end if
  if $columnNode.tStart < btTime.tStart$ **then**
    $btTime.tStart \leftarrow columnNode.tStart$
  end if
  if $columnNode.vEnd > btTime.vEnd$ **then**
    $btTime.vEnd \leftarrow columnNode.vEnd$
  end if
  if $columnNode.tEnd > btTime.tEnd$ **then**
    $btTime.tEnd \leftarrow columnNode.tEnd$
  end if
end for
return $btTime$

3.2.2 Difference

This operation mainly finds for which time interval one bitemporal element is not valid or has no transaction while another is valid or has transaction. This element could be either a table entity or a column. There can be two separate time intervals for one bitemporal element

(valid, transaction). That is, what an array of time intervals returns: The first thing is to find the correct nodes described above. After that, the algorithm works a little different from others. If a column is chosen, the conjunction is found (Algorithm 10). Then the time interval on valid times for a single entity is located to check whether there is a difference. This interval is stored as the first element of the interval array. The same thing is located as transaction time. The result is stored as the second element of the time interval array. Algorithm 11 is invoked to find this time interval array.

The difference operator is referred to in Figure 3.4; it computes and returns the time difference bwtween the first and the second time stamps. Ids and the schema must be the input; the columns are optional. If there is no column name specified, the difference operator works on the entire node with the given id. Otherwise, the conjunction is found (Algorithm 10).

According to the algorithm, relevant nodes are retrieved by using ids. ID1 is assigned to node1 and ID2 is assigned to node2. If one of the nodes is null, the none null node is assigned to the null one. If both are null, it returns null. For each node, the BitemporalTime object is instantiated with Algorithm 4. ID1 is taken as the reference. For example, if the $vStart$ value of ID1 is older than the $vStart$ value of ID2 then the returned arguments for the first element of the bitemporal time-stamp are the $vStart$ values of ID1 and ID2 in order. Otherwise, it returns null. If the $vEnd$ value of ID1 is more recent than the $vEnd$ value of ID2, the second element of the bitemporal time-stamp are $vEnd$ values of ID2 and ID1 in order. Otherwise, it returns null. These are found in the same way for $tStart$ and $tEnd$. At the end, I it is obtained that two results for each bitemporal time-stamp. In total, Algorithm 11 returns eight time-stamps. In the algorithm, the results are kept in BitemporalTime object array having two cells.

Algorithm 11: Difference($Schema$,$Table$,$Column$,$ID1$,$ID2$)

---

inputs: $Schema$ % refers to the database schema to look for

$Table$ % name of the table in the schema

$Column$ % name of the column in the table -not necessary

$ID1$ % id of the first table element

$ID2$ % id of the second table element

```
nodes ← FindNodes(Schema, Table, ID1, ID2)
node1 ← nodes[0]
node2 ← nodes[1]
btTime ← Array of two BitemporalTime objects, node1 and node2
if (node1 = null and node2 = null) then
   return null
else
   if (node2 = null) then
      node2 ← node1
   end if
   if (node1 = null) then
      node1 ← node2
   end if
end if
btTime1 ← AssignBiTemporalTime(node1)
btTime2 ← AssignBiTemporalTime(node2)
if (Column ≠ null) then
   btTime1 ← ColumnConjunction(node1, Column)
   btTime2 ← ColumnConjunction(node2, Column)
end if
if (btTime1.vStart < btTime2.vStart) then
   btTime[0].vStart ← btTime1.vStart
   btTime[0].vEnd ← btTime2.vStart
else
   btTime[0].vStart ← now
   btTime[0].vEnd ← now
end if
if (btTime1.vEnd > btTime2.vEnd) then
   btTime[0].tStart ← bitemporalTime2.vEnd
   btTime[0].tEnd ← bitemporalTime1.vEnd
else
   btTime[0].tStart = now
   btTime[0].tEnd = now
```

end if

if $(btTime1.tStart < btTime2.tStart)$ **then**

$btTime[1].Start \leftarrow btTime1.tStart$

$btTime[1].vEnd \leftarrow btTime2.tStart$

else

$btTime[1].vStart \leftarrow now$

$btTime[1].vEnd \leftarrow now$

end if

if $(btTime1.vEnd > btTime2.vEnd)$ **then**

$btTime[1].tStart \leftarrow btTime2.tEnd$

$btTime[1].tEnd \leftarrow btTime1.tEnd$

else

$btTime[1].tStart = now$

$btTime[1].tEnd = now$

end if

return $btTime$

3.2.3 Slice

The slice operator (Algorithm 12) works in the same way as the union operator, but realizes the opposite of what it does. This operation mainly finds the smallest (narrow) time boundary values for the four elements of the bitemporal time-stamp, the interval between the two bitemporal time elements. The first thing is to find the correct nodes by using *FindNodes*. In the algorithm, two ids of table rows, table, and schema must be given. The column parameter is optional. If it is null, the time-stamp comparison between rows is made, otherwise the column is considered only. In other words, this element can be found either in the entire node with sub-nodes that are uniquely identified with id value or in the specified column node. First of all, ids are given; and based on ids, related nodes are found with bitemporal time stamps. These time stamps are instantiated by invoking Algorithm 4. The slice operator compares four elements of *BiTemporal* and after the comparison, the more recent *vStart* and *tStart* values and older *vEnd* and *tEnd* values are returned. When a column is chosen, the starting and ending points of the overlaps for the column are found with (Algorithm 10).

Algorithm 12: Slice($Schema$,$Table$,$Column$, $ID1$, $ID2$)

inputs: $Schema$ % refers to the database schema to look for

$Table$ % name of the table in the schema

$Column$ % name of the column in the table -not necessary

$ID1$ % id of the first table element

$ID2$ % id of the second table element

$nodes \leftarrow FindNodes(Schema, Table, ID1, ID2)$

$node1 = nodes[0]$

$node2 = nodes[1]$

$btTime \leftarrow$ new BitemporalTime

if ($node1 = null$ and $node2 = null$) **then**

  return null

else

  if ($node2 = null$) **then**

    $node2 \leftarrow node1$

  end if

  if ($node1 = null$) **then**

    $node1 \leftarrow node2$

  end if

end if

$btTime1 \leftarrow bitemporal_{t}ime(node1)$

$btTime2 \leftarrow bitemporal_{t}ime(node2)$

if ($Column \neq null$) **then**

  $btTime1 \leftarrow ColumnUnion(node1, Column)$

  $btTime2 \leftarrow ColumnUnion(node2, Column)$

end if

if ($btTime1.vStart > btTime2.vStart$) **then**

  $btTime.vStart \leftarrow btTime1.vStart$

else

  $btTime.vStart \leftarrow btTime2.vStart$

end if

if ($btTime1.tStart > btTime2.tStart$) **then**

$btTime.tStart \leftarrow btTime1.tStart$

else

$btTime.tStart \leftarrow btTime2.tStart$

end if

if $(btTime1.vEn > btTime2.vEnd)$ **then**

$btTime.vEnd \leftarrow btTime2.vEnd$

else

$btTime.vEnd \leftarrow btTime1.vEnd$

end if

if $(btTime1.tEnd > btTime2.tEnd)$ **then**

$btTime.tEnd \leftarrow btTime2.tEnd$

else

$btTime.tvEnd \leftarrow btTime1.tEnd$

end if

return $btTime$

3.2.4 Time Queries

Time based queries can be performed with the pseudo code of the query by invoking Algorithm 13. All relevant nodes of the table are searched iteratively. According to the algorithm, if the column part criteria is specified, the $vTime$ and $tTime$ values of the column are compared at the initial phase for obtaining a result set. Here, time comparison is done with the "Between" function (Algorithm 14). In the query, $vTime$ and $tTime$ criteria are given as: If a record falls between the criteria given, it is included in the result set. When the column part is empty, then the nesting node with the id has a bitemporal time-stamp and it is compared. Otherwise, the specified columns are considered. After the result set is obtained, the projection list for display ($ProjCol$) is looked up. If exists, only the projection of the result set is displayed.

Algorithm 13: Query($Schema$,$Table$,$Column$,$VTime$,$TTime$, $ProjCol$)

```
inputs: Schema % refers to the database schema to look for
Table % name of the table in the schema
vTime % time of the valid time interval
Column % id of the first table element
tTime % time of the valid time interval
ProjCol % list of the columns to display in the table

schemaNode ← Read schemas and find the schema
tableNode ← Read tables from schemaNode and find the table
results ← newresultset
btTime ← newBitemporalTime
for tableEntity ∈ tableNode do
  if (Column ≠ null) then
    columnNodes ← ReadcolumnsfromtableNode
    for columnNode ∈ columnNodes do
      btTime ← AssignBiTemporalTime(columnNode)
      if (Between(btTime, VTime, TTime)) then
        results.add(columnNode)
      end if
    end for
  else
    btTime ← AssignBiTemporalTime(tableNode)
    if (between(btTime, VTime, TTime)) then
      results.add(tableNode)
    end if
  end if
end for
Results ← newresultset
{for projected results}
if (ProjCol ≠ null) then
  for (node ∈ results) do
    if isColumn(node) then
```

```
        Results(node.parent.ProjCol)
      else
        Results(node.ProjCol)
      end if
    end for
  else
    Results ← results
  end if
  return Results
```

According to Algorithm 14, btTime is checked regardless whether it falls in between vTime and tTime time values or not. If VTime is null then it returns null, otherwise, it returns true when vstart or btTime must be older than vTime and vEnd is more recent than vTime value. If it fails for vTime, the same comparison is repeated for tTime, and passes if it returns true.

Given btTime=[01-02-1992,15-02-2007] and [01-02-1992,15-01-2006] for [vStart,vEnd] and [tStart,tEnd] in pairs of *Day-Month-Year* format, for each entry the between function returns corresponding values for each vTime and tTime as in Table 3.2:

Return values for different time period is given in (Table 3.2).

Table 3.2: Example results of the between function

vTime	tTime	between(btTime,vTime,tTime)
01-04-2000	15-03-1999	true
11-12-2010	21-09-199	true
24-08-1998	15-01-1991	true
12-10-1988	now	false

Algorithm 14: Between(*btTime*,*VTime*,*TTime*)

```
inputs: btTime % Timestamp element
VTime % time of the valid time interval
TTime % time of the transaction time interval

between ← false
```

if $((VTime = null)$ or $((btTime.vStart \leq VTime)$ and $(btTime.vEnd \geq VTime)))$ **then**

$between \leftarrow true$

end if

if $((transactionTime = null)$ or $((btTime.tStart \leq TTime)$ and $(btTime.tEnd \geq TTime)))$

then

$between = true$

end if

return $between$

CHAPTER 4

CONCLUSION AND FUTURE WORK

Using fuzziness in XML is very easy to construct and implement. A good result occurs when easiness of XML is combined with power of fuzzy logic. Also, representation becomes very simple. It is enough to adjust a few part of the original XML and then fuzziness can be used very easily. Fuzziness makes things simple and flexible. Everyone can use an XML document and take necessary information using linguistic variables. No extra information is needed. Using fuzziness in small XML documents used in web is as practical as using it on database.

Temporal databases are used for keeping the time dimension of the data. Mostly, relational databases are used for this purpose. Temporal relational database model to XML schema conversion is proposed; the algorithm for converting temporal database content to XML format with fuzzy information is described. Existing temporal database operators are defined (union, slice, difference and query). Further, a fuzzy query algorithm has been demonstrated with results. At the end, a fuzzy partitioning scheme has been proposed for assisting end users. Yet, a system is implemented, involving complete set of methods that successfully achieves the above mentioned operations generic to any database running under MySQL or Oracle or any database system coded truly for the interface. Any bitemporal system which uses this interface can easily be adopted to the system. The methods, which are described in the Database interface, are enough to convert the database to fuzzified bitemporal XML. A new tuple can be inserted and or updated inside this XML. This system can designate fuzzy and bitemporal attributes column settings, and linguistic variables can be defined with parameters for each fuzzy column.

There are different types of XML to keep the fuzziness. In this thesis, combining all the useful

parts of each of them is tried. As a result, a schema to be used for all kind of fuzzy operations is constructed. After that this bitemporal features are added. Unless this property is used according to project, fuzzy property can still be used without looking the bitemporal parts of the project. There are some special tags to represent fuzziness. Entire tables are kept in the same file and every fuzzified table element contains ”fuzzy” attribute. All fuzzy information keeps as child nodes of this table elements within ”FUZZY” tags. The fuzzy set value and its representative value are stored inside of the tag. Fuzzy queries are processed over the fuzzified XML. That is why this is platform and program independent. Each of the results has a rank corresponding to the related query and the results are ordered by that rank. The rank also represents the fuzzy part of the system. Because there may be floating values instead of all integer values.

There is not much research about fuzziness in XML if we think usage of this technology in daily life. Thus, a convention about fuzziness in XML is not set. More articles need to be published about this topic. Everybody proposes lots of different solutions for problems. There are different fuzzy XML tools to create queries or inserting fuzzy elements to the XML files. This is quite understandable because of features of XML; however it could be good to make an aggregation on representation.

This thesis is a beginning to use fuzziness in applications, especially in time oriented areas. It is useful to be used to carry data on XML form in Web-services. Financial issues are very relevant to this topic. Financial data is temporal and large amount of data can give good and meaningful fuzzy sets. Most of this type of data are kept in relational database which is suitable to convert to XML with this system. That is why financial -or similar type of data- can be adopted to this system easily. Both, bitemporal and fuzzy operations are applicable and can give better results if they are interpreted and be constructed better. This is a very big and important topic to research. This technology is just at the start of its path. This is a very useful technology and an open area. As an example, fuzziness in XML can be adapted with ORM (Object Relational Model). Thus, fuzziness in XML and fuzzy XML databases will improve while the researches continue.

REFERENCES

[1] Tansel, Abdullah Uz, "Temporal Relational Data Model," *IEEE Transactions on Knowledge and Data Engineering,* Vol.9, No.3, pp.464-479, 1997.

[2] Ozsoyoglu, Gultekin and Richard T. Snodgrass, "Temporal and Real-Time Databases: A Survey," *IEEE Transactions on Knowledge and Data Engineering,* Vol.7, No.4, pp.513–532, 1995.

[3] Böhlen Michael H., Jensen Christian S. and Michel Scholl, "Spatio-Temporal Database Management," *International Workshop STDBM'99,* Edinburgh, Scotland, 1999.

[4] Roddick, John F. and Spiliopoulou, Myra, "A Survey of Temporal Knowledge Discovery Paradigms and Methods," *IEEE Transactions on Knowledge and Data Engineering,* Vol.14, No.4, pp.750–767, 2002.

[5] Clifford J., Croker A. and Tuzhilin A. "On Completeness of Historical Relational Query Languages," *ACM Transactions on Database Systems,* Vol.19, No.1, pp.64–116, 1994.

[6] Pissinou, N., Snodgrass, R., Elmasri, R., Mumick, I., Özsu, M., Pernici, B., Segev, A., Theodoulidis, B., and Dayal, U., "Towards An Infrastructure for Temporal Databases," *Report of An Invitational ARPA/NSF Workshop.In Sigmod Record,* pp.35-51, 1994.

[7] Snodgrass, R., Böhlen, M. H., Jensen, C. S., and Steiner, A., "Adding Transaction Time to SQL/Temporal," change proposal. ISO/IEC JTC1/SC21/WG3 DBL MAD-147r2, ANSI X3H2-96-502r1, 1996.

[8] Snodgrass, R., Boehlen, M. H., Jensen, C. S., and Steiner, A., "Adding Valid Time To SQL/Temporal," change proposal. ISO/IEC JTC1/SC21/WG3 DBL MAD-146r2, ANSI X3H2-96-501r2, 1996.

[9] Ariav, G., "A Temporally Oriented Data Model, *ACM Transactions on Database Systems,"* 11(4):499-527, 1986.

[10] Snodgrass, R., "The Temporal Query Language TQUEL," *ACM Transactions on Database Systems,* 12(2):247-298,1987.

[11] Snodgrass, R. T., Ahn, I., Ariav, G., Batory, D., Clifford, J., Dyreson, C. E., Elmasri, R., Grandi, F., Jensen, C. S., Kifer, W., Kline, N., Leung, K. K. T. Y. C., Lorentzos, N.,Roddick, J. F., Soo, A. S. M. D., and Sripada, S. M., "TSQL2 Language Specification," *SIGMOD Rec.,*, Vol.23, No.1, pp.65–86, 1994.

[12] Gadia, S., "Homogeneous Relational Model and Query Languages for Temporal Databases," *ACM Transactions on Database Systems,* 13(4):418-448, 1988.

[13] Kim, J., Yoo, H., and Lee, Y., "Design and Implementation of A Temporal Query Language With Abstract Time," *Information Systems,* 15(3):349-357, 1990.

[14] Shashi K. Gadia, ”Ben-Zvi’s Pioneering Work in Relational Temporal Databases,” *Temporal Databases*, pp.202–207, 1993.

[15] Bhargava, G. and S. Gadia, ”Achieving Zero Information Loss in a Classical Database Environment,” *Proceedings of the Conference on Very Large Databases,* pp.217-224, 1989.

[16] Tansel A. U. and. Atay C. E, ”Nested Bitemporal Relational Algebra,” *Proc. of ISCIS,* pp. 622-633, 2006.

[17] Ozsoyoglu, M. Z., Ozsoyoglu, G., ”An Extension of Relational Algebra for Summary Tables,” *Proceedings of the 2nd Intl Workshop on SDB Management,* 1983.

[18] Tansel, A. U., Garnett, L., ”Nested Historical Relations,” *Proc. ACM SIGMOD Conf.,* 1989.

[19] Ali, Khadija Abied and Pokorný, Jaroslav, ”A Comparison of XML-Based Temporal Models,” *Second International Conference on Signal-Image Technology and Internet-Based Systems,* 2006, Hammamet, Tunisia, 2006.

[20] Seo-Young Noh and Shashi K. Gadia and Shihe Ma, ”An XML-based methodology for parametric temporal database model implementation,” *Journal of Systems and Software,* 81:6, 929–948, 2008.

[21] Wang, F., Zaniolo, C., ”XBiT: An XML-based Bitemporal Data Model In Proceedings of the 23rd International Conference on Conceptual Modeling, LNCS, vol. 3288, pp 810–824, Springer, 2004.

[22] Wang F., and Zaniolo C., ”Publishing And Querying The Histories Of Archived Relational Databases In XML,” *Proceedings of the Fourth International Conference on Web Information Systems Engineering,* Rome, Italy, 2003.

[23] Noh S,. Gadia S., ”A Comparison Of Two Approaches To Utilizing XML In Parametric Databases For Temporal Data,” *Information and Software Technology,* 48:9, pp.807–819, 2006.

[24] F. Grandi and F. Mandreoli, ”The Valid Web: An XML/XSL Infrastructure for Temporal Management of Web Documents,” Advances in Information Systems, First International Conference, Lecture Notes In Computer Science, Springer, 2000.

[25] M. Gergatsoulis and Y. Stavrakas, ”Representing Changes in XML Documents using Dimensions, In The XML Database Symposium (Xsym), 2003.

[26] T. Amagasa, M. Yoshikawa, and S. Uemura. ”A Data Model for Temporal XML Documents.” *International Conference on Database and Expert Systems Application(DEXA),* 2000.

[27] Dyreson, C.E., ”Observing Transaction-time Semantics With TTXPath,” *Web Information Systems Engineering(WISE),* 2001.

[28] Zhang, S., Dyreson, C.E., ”Adding Valid Time ToXPath In: Bhalla, S. (ed.) DNIS, Lecture Notes in Computer Science, vol. 2544, pp. 29?42. Springer, Heidelberg,2002.

[29] Anthony Lo, Reda Alhajj, Ken Barker, ”VIREX:Interactive Approach for Database Querying and Integration by Re-engineering Relational Data into XML,” Web Intelligence, 1079-1080,2006

[30] Anthony Lo, Reda Alhajj, Ken Barker: ”VIREX: Visual Relational To XML Conversion Tool. J.,” Vis. Lang. Comput. 17(1): 25-45, 2006.

[31] Fernàndez, M., Kadiyska, Y., Suciu, D., Morishima, A., and Tan, W., ”SilkRoute: A Framework For Publishing Relational Data In XML,” *ACM Trans. Database Syst.* Vol 27, No: 4, 438–493, December 2002.

[32] Chengfei Liu , Millist W. Vincent , Jixue Liu,”Constraint Preserving Transformation from Relational Schema to XML Schema, *World Wide Web,* v.9 n.1, p.93-110, March 2006.

[33] Haifeng Jiang , Howard Ho , Lucian Popa , Wook-Shin Han, ”Mapping-driven XML Transformation,” Proceedings of the 16th international conference on World Wide Web, Banff, Alberta, Canada, May 08-12, 2007.

[34] Jang J. S. R. and Sun C. T., ”Neuro-Fuzzy Modeling and Control *IEEE Proceedings,* vol. 83, pp.378–406, Mar.1995.

[35] Gaurav, Abhishek and Alhajj, Reda, ”Incorporating Fuzziness in XML and Mapping Fuzzy Relational Data into Fuzzy XML,” SAC ’06: Proceedings of the 2006 ACM symposium on Applied computing, isbn :1-59593-108-2, pp. 456–460,Dijon, France, 2006.

[36] Lo, Anthony and Kianmehr, Keivan and Kaya, Mehmet and Özyer, Tansel and Alhajj, Reda, ”Wrapping VRXQuery with Self-Adaptive Fuzzy Capabilities,” WI ’07: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence,isbn :0-7695-3026-5,pp. 750–756, IEEE Computer Society, Washington, DC,USA, 2007.

[37] Alnaar Jiwani, Yasin Alimohamed, Krista Spence, Tansel Özyer, Reda Alhajj, ”Fuzzy XML Model for Representing Fuzzy Relational Databases in Fuzzy XML Format,” ICEIS (1), 163-168,2006.

[38] Oliboni, B. and Pozzani, G., ”Representing Fuzzy Information by Using XML Schema,” In Proceedings of the 2008 19th International Conference on Database and Expert Systems Application, (September 01 - 05, 2008).

[39] D. Rodrigues, A.J.O.Cruz, R.T.Cavalcante, ”A Proposal for a Fuzzy Database Architecture Incorporating XML,” *Fuzzy Sets and Systems,*160(2): 269-279, 2009

[40] Z.M. Ma, Li Yan, ”Fuzzy XML Data Modeling With the UML and Relational Data Models,” Data Knowl. Eng. 63(3): 972-996,2007.

Based on XML Applied Computing (Honolulu, 1563-1567,2009.

[41] Li Yan , Z. M. Ma , Jian Liu, ”Fuzzy Data Modeling Based on XML Schema,” Proceedings of the 2009 ACM symposium on Applied Computing, Honolulu, Hawaii, March 08-12, 2009.

[42] Jian Liu , Z. M. Ma , Li Yan, ”Efficient Processing of Twig Pattern Matching in Fuzzy XML,” Proceeding of the 18th ACM conference on Information and knowledge management, Hong Kong, China, November 02-06, 2009.

[43] Goralwalla I. A., Özsu M. T., ”Temporal Extensions to a Uniform Behavioral Object Model,” In Proceedings of the 10th International Conference on the ER Approach, pages 110 - 121, 1993.

[44] Wuu G., Dayal U., ”A Uniform Model for Temporal and Versioned Object-oriented Databases,” pp. 230–247. In A. Tansel et al., editors, Temporal Databases, Benjamin/Cummings, 1993.

[45] Käfer W., Schöning H., ”Realizing a Temporal Complex-object Data Model,” *ACM SIGMOD Record,* Vol 21, No: 2, pages 266–275, 1992.

[46] Su S., Chen H., ”A Temporal Knowledge Representation Model OSAM/T and Its Query Language OQL/T,” *International Conference on Very Large Data bases, September,* 1991.

[47] Toobis, 2005 TOOBIS Esprit Project, ”Toobis: Temporal Object-Oriented Databases within Information Systems,” Technical Report T12R.3,1999 URL <http://www.mm.di.uoa.gr/ toobis/>, Feb 23, 2010.

[48] W3C: Extensible Markup Language (XML) 1.1, 3rd edn., W3C Recommendation (February 04, 2004), URL <http://www.w3.org/TR/xml11/>

[49] Bray, Tim; Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, Franois Yergeau (September 2006). ”Extensible Markup Language (XML) 1.0 (Fourth Edition)-Origin and Goals”. World Wide Web Consortium, October 29 2006.

APPENDIX A

PROGRAM SCREENSHOTS

Some key points of the system are able to be understood from the figures below.

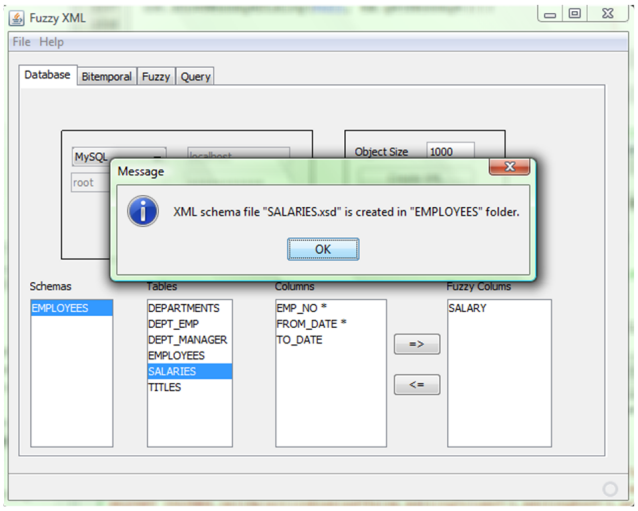

Figure A.1: XML Schema Definition

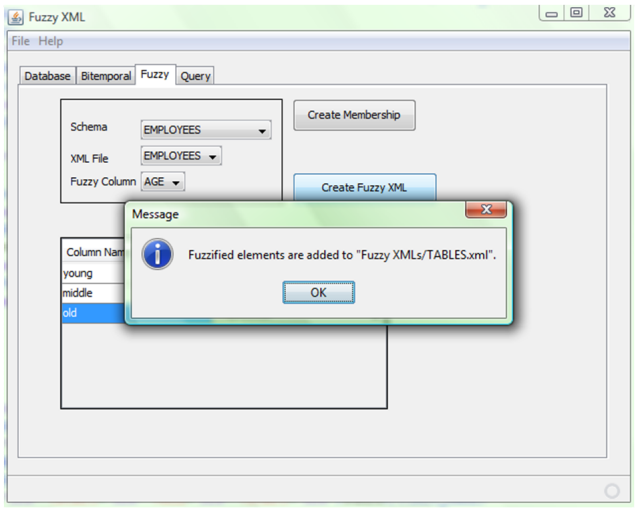

Figure A.2: Incorporating Fuzziness in XML

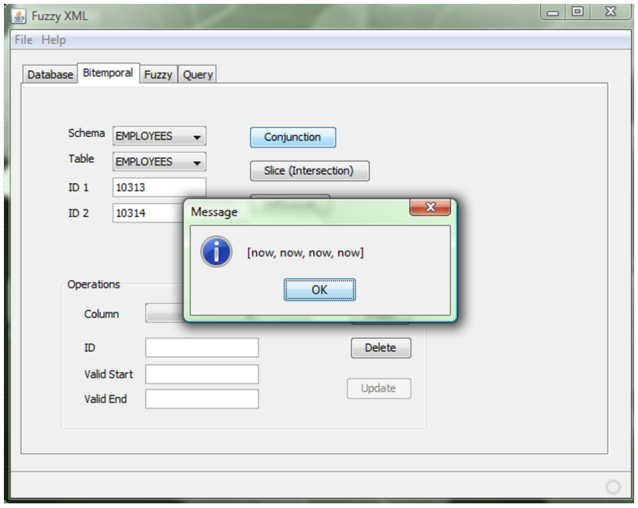

Figure A.3: Temporal Operators in XML

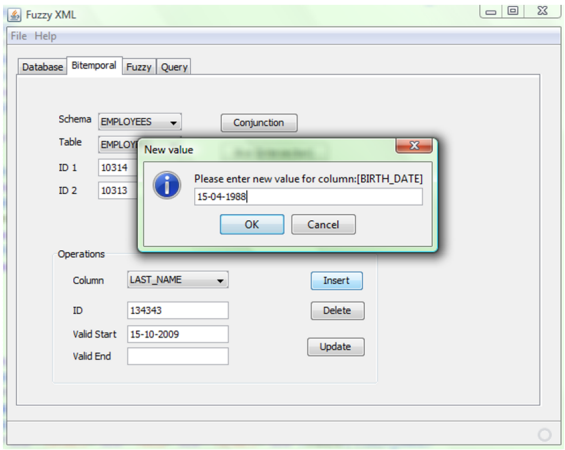

Figure A.4: Data Manipulation in XML

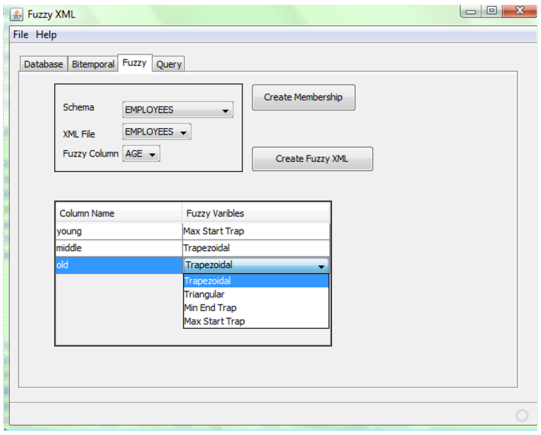

Figure A.5: Fuzzy Column Definition with Parameters

APPENDIX B

CONFIGURATION FILE

<?xml version=”1.0” encoding=”UTF-8”?>
<CONFIG>
<SCHEMA name=”EMPLOYEES”>
<TABLE name=”EMPLOYEES”>
<COLUMN name=”AGE”>
<FUZZY_VALUES>
<FUZZY end=”49.0” ignore=”start” maxEnd=”45.0” maxStart=”45.0” name=”young” start=”45.0”/>
<FUZZY end=”53.0” maxEnd=”49.0” maxStart=”47.0” name=”middle” start=”47.0”/>
<FUZZY end=”54.0” ignore=”end” maxEnd=”51.0” maxStart=”51.0” name=”old” start=”51.0”/>
</FUZZY_VALUES>
</COLUMN>
</TABLE>
<TABLE name=”SALARIES”>
<COLUMN name=”SALARY”>
<FUZZY_VALUES>
<FUZZY end=”55586.0” ignore=”start” maxEnd=”39507.0” maxStart=”39507.0” name=”low” start=”39507.0”/>
<FUZZY end=”62607.0” maxEnd=”47883.0” maxStart=”47883.0” name=”average” start=”47883.0”/>
<FUZZY end=”70495.0” ignore=”end” maxEnd=”55685.0” maxStart=”55685.0” name=”high” start=”55685.0”/>
</FUZZY_VALUES>
</COLUMN>
</TABLE>
<TABLE name=”SALARIES”>

<COLUMN name="SALARY">
<FUZZY_VALUES>
</FUZZY_VALUES>
</COLUMN>
</TABLE>
</SCHEMA>
<SCHEMA name="FINANCE">
<TABLE name="QQQQ_MAN">
<COLUMN name="VOLUME">
<FUZZY_VALUES>
<FUZZY end="7.4045536E7" maxEnd="5079933.5" maxStart="5079933.5" name="low" start="5079933.5"/>
<FUZZY end="1.05222744E8" maxEnd="7.40622E7" maxStart="4.7986368E7" name="average" start="4.7986368E7"/>
<FUZZY end="1.31139072E8" ignore="start" maxEnd="8.8562328E7" maxStart="8.8562328E7" name="dangerous" start="8.8562328E7"/>
</FUZZY_VALUES>
</COLUMN>
<COLUMN name="LOW">
<FUZZY_VALUES>
</FUZZY_VALUES>
</COLUMN>
</TABLE>
</SCHEMA>
</CONFIG>

APPENDIX C

DATABASE INTERFACE

```
package fuzzyxml.database;
import fuzzyxml.items.ComboItem;
import java.util.List;

/**
*
* @author Ömer Özgün IŞIKMAN
* @version 2.0
* Interface to get values from different types of databases
*/
public interface Database {

//List schemas for current database
public List<ComboItem>getSchemas();
//List tables for selected schema
public List<ComboItem>getTables(String schema);
//Get the size of the table
public List<ComboItem>getColumns(String schema,String table);
//Get the columns for the selected table
public int getTableSize(String schema,String table);
//Get the values of the selected table from the database
public List getTableResult(String schema, String table, int size, byte chooseType);
}
```