MANAGEMENT OF TEMPORAL EVENTS FOR FUZZY XML DATABASES

Sangeetha Veerappan
B.S., PSG College of Technology, India, 2006

PROJECT

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2009
MANAGEMENT OF TEMPORAL EVENTS FOR FUZZY XML DATABASES

A Project

by

Sangeetha Veerappan

Approved by:

______________________________, Committee Chair
Dr. Ying Jin

______________________________, Second Reader
Dr. Cui Zhang

______________________________
Date
Student: Sangeetha Veerappan

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the Project.

__________________________, Graduate Coordinator    ________________
Dr. Cui Zhang                    Date

Department of Computer Science
Abstract

of

MANAGEMENT OF TEMPORAL EVENTS FOR FUZZY XML DATABASES

by

Sangeetha Veerappan

Databases based on relational models support crisp and precise data. In the real world scenario, there exists a lot of uncertainty and imprecise information. Fuzzy logic breaks the rigid crisp logic boundaries and allows decisions to be taken in a more realistic manner. In this project, I have implemented a system supporting mutating events and temporal events using fuzzy active rules over XML database systems.

The project included a language definition and an execution environment of timer rules in fuzzy active format for XML Databases. The language definition includes a grammar to specify timer rules, allowing fuzzy expressions in the condition part of a rule. The rule engine was developed to support the execution of rules with different types of fuzzy distributions. The project also designed and implemented the fuzzy XML system allowing fuzziness in database storage and queries. The project analyzed the system performance by comparing the execution overhead on fuzzy queries over crisp queries.

_____________________, Committee Chair
Dr. Ying Jin

_____________________
Date
I would like to express my greatest gratitude to the people who have helped and supported me throughout this project work.

First and foremost, I would like to thank my project advisor, Dr. Ying Jin for all her valuable guidance and advice. She constantly supported me during the entire life cycle of this project. Dr. Ying Jin guided my study on fuzzy database and technologies during the early research study for this project and she was always there to provide the suggestions and assistance during design and implementation of this project. I am also grateful for her assistance with flexibility which provided me an option to continue my internship along with this project work. I express my honest gratitude and appreciation for her continuous support through this project.

I would like to thank Dr. Cui Zhang for her support towards completion of this project. Also, I would like to take this opportunity to thank California State University Sacramento, for providing me with this opportunity to participate and learn in a good environment provided with pleasant facilities.

Lastly, I thank Chandrashekar J Madalli for helping me in person and through emails, to understand the fuzzy concepts that I needed for this project. My sincere thanks to all my friends who provided me with their advice and help. I am grateful to my parents and family members for their constant support and encouragement in my endeavor to complete this project.
# TABLE OF CONTENTS

| Acknowledgments                              | v     |
| List of Tables                                | ix    |
| List of Figures                               | x     |
| 1 INTRODUCTION                               | 1     |
| 1.1 Databases - Background Information        | 1     |
| 1.2 Fuzzy Logic                              | 1     |
| 1.3 Project Purpose                          | 2     |
| 1.4 Organization                             | 3     |
| 2 RELATED STUDY                               | 4     |
| 2.1 Fuzzy Concepts Overview                  | 4     |
| 2.2 Representation of Knowledge              | 5     |
| 2.2.1 Linguistic Label With Possibility Distribution | 6 |
| 2.2.2 Linguistic Label With Possibility Interval | 7 |
| 2.2.3 Approximate Values                     | 8     |
| 2.3 Relational Comparison of Crisp and Fuzzy Values | 8 |
| 2.4 Temporal Events and Timers               | 9     |
| 2.5 Tools and Technology                     | 10    |
| 2.5.1 XML                                    | 10    |
| 2.5.2 Oracle Berkeley XML Database           | 11    |
2.5.3 XQuery .................................................................11
2.5.4 JAVA CC Complier ..................................................12
3 SYSTEM DESIGN ..........................................................13
  3.1 System Architecture ..................................................13
  3.2 Temporal Events Handling ..........................................14
4 SYSTEM IMPLEMENTATION .............................................17
  4.1 Bookstore Application ...............................................17
  4.2 Fuzzy Active Rules ..................................................19
    4.2.1 Representation of Fuzzy Rule ..................................19
    4.2.2 Fuzzy Parser ...................................................23
    4.2.3 Fuzzy to Crisp Translation ....................................23
  4.3 Fuzzy Terms in Database ..........................................28
  4.4 Timer Rules ..........................................................29
    4.4.1 Timer Rules Processing ........................................32
5 SYSTEM QUERYING AND PERFORMANCE ANALYSIS ............35
  5.1 Database Query .....................................................35
  5.2 Performance Analysis ..............................................36
6 TEST PLAN AND RESULTS .............................................39
  6.1 Validation of Fuzzy Parser ........................................39
  6.2 Validation of Crisp Rule ...........................................41
  6.3 Validation of Querying ............................................41
LIST OF TABLES

Table 4.1 Translation Parameters for a Crisp Term in Database ......................... 25
Table 4.2 Translation parameters for Fuzzy Term in Database ............................ 26
Table 5.1 Performance Analyses Report ............................................................ 37
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Probability Density Function of Trapezoidal Distribution</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Calculation of Min and Max Using Trapezoidal Distribution</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Probability Distribution for Interval Possibility Distribution</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>Probability Distribution for Triangular Possibility Distribution</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>System Architecture to Convert Fuzzy Rule to Crisp Rule</td>
<td>14</td>
</tr>
<tr>
<td>3.2</td>
<td>System Design for Temporal Events Processing</td>
<td>15</td>
</tr>
<tr>
<td>4.1</td>
<td>Linguistic Variables and Values</td>
<td>18</td>
</tr>
<tr>
<td>4.2</td>
<td>Fuzzy Rule Templates</td>
<td>20</td>
</tr>
<tr>
<td>4.3</td>
<td>An Example Fuzzy Active Rule</td>
<td>21</td>
</tr>
<tr>
<td>4.4</td>
<td>Crisp Rule Translation Output</td>
<td>27</td>
</tr>
<tr>
<td>4.5</td>
<td>Template for Timer Rule</td>
<td>30</td>
</tr>
<tr>
<td>4.6</td>
<td>An Example Timer Rule</td>
<td>30</td>
</tr>
<tr>
<td>4.7</td>
<td>Template to Drop a Timer Rule</td>
<td>31</td>
</tr>
<tr>
<td>4.8</td>
<td>Cancellation of Temporal Event</td>
<td>33</td>
</tr>
<tr>
<td>4.9</td>
<td>Temporal Event Execution Result</td>
<td>34</td>
</tr>
<tr>
<td>5.1</td>
<td>Simple Query</td>
<td>35</td>
</tr>
<tr>
<td>5.2</td>
<td>Query Execution Time</td>
<td>37</td>
</tr>
<tr>
<td>5.4</td>
<td>Performance Analysis Chart for Complex Queries</td>
<td>38</td>
</tr>
<tr>
<td>6.1</td>
<td>Exception on an Invalid Character to Represent Fuzzy Term</td>
<td>39</td>
</tr>
</tbody>
</table>
Figure 6.2 Exception for Missing Term in a Rule ................................. 40
Figure 6.3 Error for Missing Node ..................................................... 40
Figure 6.4 Testing Insertion of Nodes in DB ...................................... 42
Figure 6.5 Correctness of Database Fuzzy Term Interpretation ............. 43
1.1 Databases - Background Information

Origin of databases can be routed back to the need to store huge data sets for business and medical purposes. Databases have been vigorously studied since Codd (1970) proposed the relational model for data management [1]. Such database systems can deal with sample sets which contain precise numerical data. In the real world, there exists a lot of uncertain and ambiguous data, which cannot be defined precisely for storage in traditional crisp storage databases. In the current day systems, the need to store vague terms becomes a needy aspect. This need evolved the concept of fuzziness in the database world.

An active database is a database that includes active rules, mostly in the form of ECA rules. The event part specifies the triggering signal, and condition states the reasoning. If the condition results to be true, then the action part is executed. ECA rules allow users to specify business logic and automate reaction to events [2, 3].

1.2 Fuzzy Logic

Human beings possess an inborn talent to naturally evaluate multiple parameters and process all sorts of data. The concluding power of the human brain needs no more of an additional processing to analyze any situation. For example, the approximation involved in judging the speed of a car entering from a side street to a highway and its
probability to pull up in front of another car is totally fuzzy. To model all of this fuzzy behavior in computing systems, the concept of fuzzy logic was born. “As the complexity of a system increases, it becomes more difficult and eventually impossible to make a precise statement about its behavior, eventually arriving at a point of complexity where the fuzzy logic method born in humans is the only way to get at the problem” [4].

1.3 Project Purpose

Databases based on relational models support crisp and precise data. In the real world scenario, there exists a lot of uncertainty and imprecise information. Fuzzy logic breaks the rigid crisp logic boundaries and allows decisions to be taken in a more realistic manner. In this project, I have implemented a system for the generation of temporal events using timer rules for fuzzy XML Databases, which enable execution of events in an asynchronous manner and enhance the speed and performance to high scales when compared to traditional mutating events. This work is in continuation of the rule system designed and implemented in [13] that includes a rule repository, event handler and rule engine that allows multiple rules to be executed based on priorities to fulfill the business logic defined by rules. Enhanced features have been added to the existing system which includes several distributions being used to define membership functions in fuzzy active rules, allowing fuzzy terms storage in the database, and inclusion of fuzzy expressions in queries.

The project also includes language definition and automization of timer rules in fuzzy active format for XML Databases. The language definition includes a grammar to specify timer rules, allowing fuzzy expressions in the condition part of a rule. A parser is
being implemented to take as input fuzzy active rules and scheduling information and will setup the event to run against the XML database. The rule engine in [13] is enhanced to support triggers and rules with different distributions and introduce fuzziness in storage and queries. The corresponding changes to the parser are also implemented. The entire system was tested to take in fuzzy rules with various linguistic variables, and timer rules were scheduled to run in a dynamic setup.

1.4 Organization

The rest of the report is organized as follows. Chapter 2 introduces and presents the related work, technology and tools used in the project. Chapter 3 covers the system design involved in crisp rule generation, database fuzzy terms handling and interpretation, and generation and management of timer rules for fuzzy xml databases. Chapter 4 presents the system implementation details. Chapter 5 showcases the database querying methodology, and performance results of crisp and fuzzy queries execution against the database. Chapter 6 covers the various test cases used to test the application. Chapter 7 concludes the report with project summary and presents future work.
2.1 Fuzzy Concepts Overview

Traditional databases operate on large samples of precisely defined data. In real world, a lot of uncertainty exists which creates a need to represent imprecise information. Fuzzy logic breaks the rigid crisp boundaries and allows decisions to be taken in a more realistic manner. The concept was conceived by Zadeh in 1964 when he was trying to contemplate how to program software for handwriting recognition [5].

Human beings manipulate decisions based on pieces of information and generally reason them with vague terms like “The boy is young”, or “He is running too fast”. The manipulation needs additional processing on the computers part. The term “young” would cause a lot of errors when linked with relational databases which are tuned to incorporate numerical values. Fuzzy logic extends computer capabilities to deal with imprecise and vague information.

Inconsistency, imprecision, vagueness, uncertainty, and ambiguity are five kinds of imperfect information in databases [6]. Inconsistency explains a property where the same attribute may hold different values. An example would be a person who may maintain multiple statuses of both a student and a part time employee. Imprecision and vagueness relate to attribute values picked from a range set rather than a defined numerical content. As an example, ‘young’ relates to a value in a numerical range like [10, 25]. Uncertainty
is the degree of truth of an attribute value. Ambiguity leads to several interpretations of the content. Furthermore the different kinds of imperfect information can co-exist with each other [6]. Fuzzy set theory [5] is a generalization of the crisp sets and proposes methods to resolve the different forms of information representation.

The membership function in a fuzzy set is not a matter of true or false but a matter of degree [5]. Consider an example, “The temperature is hot”. The temperature is called a fuzzy variable or a linguistic variable, which takes values as words or sentences based on linguistic values (hot), which are in turn connected to numerical values through membership functions. The different distributions used to define membership functions are described in the next section “Representation of knowledge”.

2.2 Representation of Knowledge

In this section, the knowledge representation for the different information types are presented. Every linguistic variable is decomposed into a set of linguistic values. Linguistic variables along with the set of values which describe it together form the Fuzzy Metaknowledge Base (FMB). FMB takes the responsibility of storing all the necessary information related to the imprecise data representation for the attributes [8].

Types of Knowledge representation include the following [8]:

a. Linguistic label with possibility distribution

b. Linguistic label with possibility interval
c. Approximate values

**2.2.1 Linguistic Label With Possibility Distribution**

The Linguistic variable in this case includes values which are of the form \([\alpha, \beta, \gamma, \delta]\). The numerical value for the variable is computed using trapezoidal distribution. The probability density function of the trapezoidal curve taken from [8] is shown in Figure 2.1.

![Figure 2.1 Probability Density Function of Trapezoidal Distribution [8]](image)

Trapezoidal distribution enables the membership function to calculate the numerical value with a threshold between 0 and 1. Based on the threshold value, the min and max are calculated using the formulae [13]:

$$
Min = (\beta - a) \ast THOLD + a
$$

$$
Max = [(\delta - \gamma) \ast (1 - THOLD)] + \gamma
$$

As an example, we consider the graph in Figure 2.2 from [13] to illustrate min and max calculation using trapezoidal distribution.
Using the formulae, the min value is calculated as $\text{min} = 30$ and $\text{max} = 50$. When $\text{threshold} = 0.0$, min equals the value of $\alpha$ and max equals the value of $\delta$.

### 2.2.2 Linguistic Label With Possibility Interval

The Linguistic variable has its value from an interval $[m, n]$ and the corresponding mathematical distribution is interval possibility distribution. Figure 2.3 shows the probability distribution of an interval possibility distribution as taken from [8].
The range m and n denote the min and max for any value of threshold between 0.0 and 1.0.

2.2.3 Approximate Values

Linguistic variable, whose value is a vague concept like ‘approximately d’, is defined by a triangular possibility distribution as shown in the Figure 2.4 from [8].

![Triangular Possibility Distribution](image)

**Figure 2.4 Probability Distribution for Triangular Possibility Distribution [8]**

Triangular distribution takes a median value (d) and a margin value (m) with the threshold (THRES) between 0.0 and 1.0. The min and max are computed using the formulae:

\[
Min = d - (margin \ast (1 - THRES))
\]

\[
Max = d + (margin \ast (1 - THRES))
\]

2.3 Relational Comparison of Crisp and Fuzzy Values

In conventional relational database systems, since an attribute holds precise numerical values, a query against a database depends on whether the tuple satisfies the
condition or not. As a contrast in fuzzy databases, tuple satisfaction is a range between 0 and 1. Tuples with a degree greater than 0 are presented in the result set to the user [9].

Considering the degree of comparison, three cases are possible [9]:

a. **Both values are crisp**

   This relates to a normal numerical comparison of values as found in relational systems. If the degree of satisfaction is 1, the tuple is part of the output.

b. **One value is crisp and the other one being fuzzy**

   Fuzzy terms are represented by one of the membership functions as described in the previous section (triangular, trapezoidal or interval). If the crisp value which is denoted by a straight line intersects with the membership function at a point greater than 0 in the first quadrant, it causes the tuple to be part of the result set.

c. **Both values are fuzzy**

   Fuzzy terms in this case consist of two membership functions. The degree of satisfaction would be the point of intersection of the two curves.

2.4 Temporal Events and Timers

Traditional databases can handle events such as insert, update and delete. With regards to scalability, the DML statements are not self sufficient to incorporate automatic
reaction to real world applications. “Temporal events are the ones generated based on a
certain time or time interval” [10]. The concept of timer trigger has been proposed by
[11] to allow condition of a trigger to be checked less frequently than traditional triggers.

An example of a temporal event is, “Check inventory at 12:00pm everyday”. The
event is scheduled to run at 12:00pm with the frequency of recurrence being 24 hours.
Events automated using timers exponentially increase efficiency of operation.

The concept of timers in Java is used to enable temporal events. Timers enable a
specific task to be performed repeatedly with a certain delay. Each timer has one or more
action listeners and a delay which denotes the time in between recurring events.

2.5 Tools and Technology

2.5.1 XML

XML is a text based markup language which is more than html [13]. Enhanced
readability, powerfulness to express hierarchical relationships among entities, being
language and OS independent are some of the advantages of using XML. XML is
classified as an extensible language, because it allows the user to define the mark-up
elements. It allows the end user to create their own mark-up elements to support their
application which is very different like the fixed tags in HTML. The main purpose of
HTML is to format the data that are presented through browser for an application. The
purpose of XML is not to format data but to store the data for the application [13]. DOM
parsers are used to create a memory map for XML data. In this project the entire application, the rules and templates are designed in XML.

**2.5.2 Oracle Berkeley XML Database**

The Oracle Berkley XML database is an open-source, embedded, native XML database which is built on top of the Oracle Berkley database. Apart from sharing all the rich features of the Oracle Berkley database, Oracle Berkeley DB XML adds a document parser and Xquery engine to enable the fast, most efficient retrieval of data. The official Berkley distribution provides rich set of core libraries for C++, Java, Perl, Python, PHP, and Tcl languages. Oracle Berkley XML is supported on a very large number of platforms in which we can link many applications using the wide variety of languages supported by Oracle Berkley XML [14].

**2.5.3 XQuery**

Xquery is the language to query XML data, similar to the usage of SQL for relational databases. XQuery is built on XPath and is a W3C recommendation. XQuery 1.0 is standardized and highly used in the current systems. Xquery provides needed concepts, upgraded functionality, and new ideas that will fundamentally change the way XML applications are designed and implemented. The distinguishing factor for Xquery as compared to other XML query language is its accomplishments of its features which are only dedicated to serve for querying the XML data. Xquery has robust functions that can perform the database joins and sorts effectively and can be used to manipulate the
sequence of resulting data from XML nodes in a simplified way. We can say that as SQL is used for querying relational data, Xquery is used to query XML data [13, 15].

2.5.4 JAVA CC Compiler

JavaCC or Java Compiler Compiler is a widely used tool for lexical and parser component generation which follows regular expression and BNF (Backus-Naur Form) notation syntax for lexical specifications. JavaCC reads a formal grammar in BNF notation and converts it into java code for a parser. BNF is a metasyntax notation used to express context-free grammars [16]. For more information on the tools described in this section, please refer to [13].
Chapter 3

SYSTEM DESIGN

This section provides the system design of the rule system that translates a fuzzy rule to crisp rule and the enhancement to introduce fuzziness in storage and queries. Several different distributions are incorporated to represent the membership functions. The template design for specification and automization of timer rules is also presented.

3.1 System Architecture

The system architecture involved in converting fuzzy active rules to crisp rules is presented in this section. The entire flow involved in converting fuzzy rule to a crisp format is shown in the Figure 3.1. The architecture involves user interface, fuzzy parser, and XML databases to store fuzzy and crisp rules.

In this section, we first start with the detailed description of each of the components involved and then understand how each of the modules interacts with each other. The user inputs rules and queries in fuzzy format through the user interface to the system. The fuzzy rules being inputted are stored in a repository (frule.xml). The Fuzzy parser interprets the fuzzy components in the rules based on the grammar and linguistic values (fuzzy.xml), and converts it to its crisp equivalent. These converted crisp active rules are stored in another repository (crule.xml).
3.2 Temporal Events Handling

The entire flow involved in handling temporal events through timer rules is shown in Figure 3.2. There are two cycles involved in handling temporal events. The first one enables the user to add new timer rules. The rules get added to a repository (temporal_events.xml).
The other flow involves parsing the timer rules from the XML repository and scheduling the rules. The entire action is automated through a master cron job which runs in a time interval to identify rules which have been newly added to the rules database and performs the needed action. If the status of the timer rule is “new”, the timer is scheduled based on the rule parameters and its status is made active. On the other hand, a rule with status “drop”, cancels the timer scheduled based on the rulename. The rule with a status
“active” runs uninterrupted. The language involved in representing timer rules is described in the next chapter.
Chapter 4

SYSTEM IMPLEMENTATION

This chapter presents the Bookstore application used to illustrate our approach. We also discuss the implementation details of specifying rules using different distributions, incorporating fuzziness in database, and the rule engine to process timer rules.

4.1 Bookstore Application

This project uses a Bookstore application to illustrate all the flows and components involved. The Bookstore database is a XML database (FuzzyData.xml) which consists of a series of nodes related to Book, Discount_information, Purchase_Order, Sales_Record and Book_Sales_History, which is based on the design in [13]. The detailed tree structure can be found in Appendix A. The application allows storage of fuzzy values in the database.

One of the major enhancements features in this project was to allow different distributions in the event and timer rules. The linguistic variables in the Bookstore application are quantity_in_stock and popularity. The three different distributions discussed in Chapter 2 are implemented for the bookstore application. The set of details to define any distribution is stored in a file which specifies the required parameters for usage. For example, a trapezoidal distribution would need α, β, γ, and δ values and for a triangular distribution, the margin value should be known. The set of Linguistic variables along with the required values for the different possibility distributions are present in a fuzzy.xml file as shown in the Figure 4.1
<FuzzyValues>

<LinguisticVariable name="quantity_in_stock">

<Type T="trapezoidal">

<LinguisticTerm name="LOW">
  <alpha>10</alpha>
  <beta>500</beta>
  <gamma>1000</gamma>
  <delta>1500</delta>
</LinguisticTerm>

<LinguisticTerm name="MED">
  <alpha>1000</alpha>
  <beta>1500</beta>
  <gamma>2000</gamma>
  <delta>2500</delta>
</LinguisticTerm>

</Type>

<Type T="approximate">
  <margin m="50" />
</Type>

</LinguisticVariable>

</FuzzyValues>

Figure 4.1 Linguistic Variables and Values
As shown in Figure 4.1, each Linguistic variable can take in different linguistic types with its set of linguistic values calculated based in its corresponding possibility distribution (trapezoidal, or triangular).

**4.2 Fuzzy Active Rules**

The rule engine designed in [13] is used to specify and process fuzzy active rules using trapezoidal distribution. The semantics involved in parsing rules with different distributions is discussed in this section.

**4.2.1 Representation of Fuzzy Rule**

A fuzzy rule is defined using a XML format as shown in Figure 4.2[13]. A fuzzy rule is uniquely identified by a rule name. A fuzzy active rule consists of two parts: an event, and a combination of condition and action. An event consists of four components as shown in four tags. `<event_type>` specifies what type of XQuery command causes the rule to be triggered: insert, delete, or replace. `<event_happen>` specifies whether the condition and action part of the rule should be executed *before* the actual XQuery command (e.g. replace) or *after* the command. `<event_node>` specifies the document to be modified. `<doc>` specifies the newly updated values. When one event triggers multiple rules, rules are executed sequentially, according to the *priority* (as shown in `<priority>`) defined by the rule writer. No two rules are allowed to have the same priority when they are triggered by the same event with the same *event_happen* value. A fuzzy rule added should have a unique *rulename*. The *condaction* part should be a valid xQuery expression.
<rules>

<rule>

<rulename>name of rule</rulename>

<event_type>insert/ delete/ replace</event_type>

<event_happen>After or Before event</event_happen>

<event_node>DOC NODE BEING REFERNCED</event_node>

<priority>priorty of rule</priority>

<doc>updated doc</doc>

<condaction>Condition and Action</condaction>

</rule>

</rules>

Figure 4.2 Fuzzy Rule Templates [13]

Figure 4.3 shows an example of a fuzzy active rule. This rule specifies an “insert” event and “after” an event. This rule is triggered and executed after insertion of a Sales Record that indicates that books were sold. The condaction specifies that if the quantity_in_stock is approximately 1200 with threshold 0.6 and popularity is high with threshold 0.6, then a place a purchase order with the same quantity as the quantity sold.
Figure 4.3 An Example Fuzzy Active Rule
The semantics part of the fuzzy active rule allows two special characters to be used. One of it is the “%”, which identifies a linguistic variable. The other one is “#”, which identifies the binding element between the event and condition. The binding element represents the variable passed between the event and condition.

A fuzzy rule could have the following symbols in the condition (condition and action) part of the template:

a. The special character ‘#’ to specify the binding element. For the rule in Figure 4.3, the binding element is newdoc which is the newly inserted Sales_Record.

b. The special character ‘%’ identifies occurrence of a linguistic variable. %quantity_in_stock and %popularity are the linguistic variables according to Figure 4.3.

c. The linguistic variable is followed by one of the relational operators (<, <=, >, >=, =, !=).

d. The relational operator should be followed by one of the linguistic terms. If a linguistic variable follows a trapezoidal distribution, then the linguistic term is to be given after the operator like %quantity_in_stock < HIGH. The values of α, β, γ, and δ for the linguistic term (HIGH) is taken from the fuzzy.xml file and the corresponding min and max value are calculated. If an approximate relationship (triangular distribution) is to be expressed, it can be done by “~” followed by the linguistic term as in %popularity = ~40. For an interval distribution, the range is given as [min, max].
4.2.2 Fuzzy Parser

A fuzzy parser is a form of a string analyzer which acts from left to right on the input and identifies fuzzy terms based on the language. Fuzzy parser is developed using JavaCC which is both a parser generator and a lexical analyzer. Lexical analyzer breaks the input into a series of tokens and then identifies the type of the token [16]. In our implementation, the series of valid tokens are presented in Appendix B.

Fuzzy parser understands a certain grammar and expects the user input to be in accordance with it. It considers a set of tokens as valid characters and works on a set of patterns to interpret the input. When a rule is parsed, the fuzzy parser scans for the special character “#” to reference an the binding element. The token “%”, scans and identifies the linguistic variable to the LHS of the operator and linguistic term on the RHS with a threshold value. The result crisp value is calculated corresponding to the distribution and its formulae to calculate min and max values as explained in Chapter 2. The code snippet of the fuzzy parser is presented in Appendix C.

4.2.3 Fuzzy to Crisp Translation

Fuzzy parser in this project scans through the input on a character basis. If a character matches one of the tokens in the valid character set, the parser continues looping. If not, a parser error would be generated with the message, “Invalid Token”. Parser loops into a separate routine on identifying a %, which denotes that the following term is the linguistic variable as the one below:
%quantity_in_stock < HIGH with threshold 0.6 (or)

%quantity_in_stock = ~LOW with threshold 0.8

Fuzzy parser identifies the linguistic variable following %. It then scans to understand the relational operator specified in the expression. The linguistic term is parsed and stored. The values of $\alpha$, $\beta$, $\gamma$, and $\delta$ are obtained from the fuzzy.xml file under the parent node matching the parsed linguistic variable and linguistic term for a variable whose membership function is trapezoidal. On the other hand, if the operator is followed by a “~”, the value of margin under type approximate for the linguistic variable is found and stored. The parser continues scanning and obtains the threshold values.

Table 4.1 shows the calculation and comparison translations for each of the linguistic terms which have crisp values in the database. The Fuzzy_term represents the linguistic variable identified from the fuzzy active rule. The RHS of the operator is one of the linguistic terms acceptable for the variable. Based on the mathematical distribution involved to map the input to its membership function, the min and max value is calculated with its corresponding formulae. The comparison translation column shows the output that will be part of the crisp rule.

Consider as an example, quantity_in_stock < ~1200 with threshold 0.6. The Linguistic_variable is quantity_in_stock and the operator is “<”. Based on the min value calculated, the crisp rule is quantity_in_stock < min.
<table>
<thead>
<tr>
<th>Linguistic variable comparison</th>
<th>Comparison translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linguistic_variable &lt; linguistic_term</td>
<td>Linguistic_variable &lt; min</td>
</tr>
<tr>
<td>Linguistic_variable &lt;= linguistic_term</td>
<td>Linguistic_variable &lt;= min</td>
</tr>
<tr>
<td>Linguistic_variable &gt; linguistic_term</td>
<td>Linguistic_variable &gt; max</td>
</tr>
<tr>
<td>Linguistic_variable &gt;= linguistic_term</td>
<td>Linguistic_variable &gt;= max</td>
</tr>
<tr>
<td>Linguistic_variable = linguistic_term</td>
<td>Linguistic_variable &gt;= min</td>
</tr>
<tr>
<td></td>
<td>Linguistic_variable &lt;= max</td>
</tr>
<tr>
<td>Linguistic_variable != linguistic_term</td>
<td>Linguistic_variable &lt; min</td>
</tr>
<tr>
<td></td>
<td>Linguistic_variable &gt; max</td>
</tr>
</tbody>
</table>

Table 4.1 Translation Parameters for a Crisp Term in Database

The translation performed if the term in the database was fuzzy is presented in Table 4.2. The min and max value corresponding to the fuzzy linguistic variable is min_f and max_f. The min and max is for the linguistic term to the RHS of the operator. If the degree of intersection of the two curves (linguistic variable, and linguistic term) is greater than zero, then the translation is satisfied.
<table>
<thead>
<tr>
<th><strong>Linguistic Variable</strong></th>
<th><strong>Comparison translation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Linguistic_variable &lt; linguistic_term</td>
<td>min_f &lt; min</td>
</tr>
<tr>
<td>Linguistic_variable &lt;= linguistic_term</td>
<td>min_f &lt;= min</td>
</tr>
<tr>
<td>Linguistic_variable &gt; linguistic_term</td>
<td>max_f &gt; max</td>
</tr>
<tr>
<td>Linguistic_variable &gt;= linguistic_term</td>
<td>max_f &gt;= max</td>
</tr>
<tr>
<td>Linguistic_variable = linguistic_term</td>
<td>(min_f &gt;= min &amp;&amp; max_f &lt;= max)</td>
</tr>
<tr>
<td>(the intersection of the two curves should be at a point greater than zero)</td>
<td></td>
</tr>
<tr>
<td>Linguistic_variable!= linguistic_term</td>
<td>(min_f &gt; max &amp;&amp; max_f &gt; max)</td>
</tr>
<tr>
<td>(there is no degree of satisfaction for the two curves)</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.2 Translation parameters for Fuzzy Term in Database**

Using the above rules for translation, the crisp rule is generated for the user input fuzzy rule and stored in the crisp rule repository.
Fuzzy translation of the rule in Figure 4.3 produces a crisp rule as shown in the Figure 4.4.

```
<rule>
  <rulename>Insert Purchase Order</rulename>
  <event_type>insert</event_type>
  <event_happen>after</event_happen>
  <event_node>Sales_Record</event_node>
  <priority>2</priority>
  <doc>newdoc</doc>
  <condaction>
    let $a := collection("data.dbxml")/Bookstore
    let $c := collection("data.dbxml")/Sales_Record return for $b in
    ($a/Book[book_ID/text() = $c/book_ID/text()]) return
    if($b/((($b/quantity_in_stock/text() >=1190.0) and
    ($b/quantity_in_stock/text() <= 1240.0)) and
    (($b/popularity_min/text() >=560.0)and($b/popularity_max/text() <=860.0)) or
    (($b/popularity_min/text() <=560.0)and($b/popularity_max/text() >=560.0))and($b/popularity_max/text() <=860.0)) and
    ($b/quantity_in_stock/text() <= 1240.0)) and
    ((($b/popularity_min/text() >=560.0)and($b/popularity_max/text() <=860.0)) or
    (($b/popularity_min/text() <=560.0)and($b/popularity_max/text() >=560.0))and($b/popularity_max/text() <=860.0)) or
    (($b/popularity_min/text() >=560.0)and($b/popularity_max/text() <=860.0)) or
    (($b/popularity_min/text() <=560.0)and($b/popularity_max/text() >=560.0))and($b/popularity_max/text() <=860.0))
    then insert nodes <Purchase_Order>
    <quantity_of_order>{$c/quantity_sold/text()}</quantity_of_order>
    <order_filled>No</order_filled> </Purchase_Order> after
    $a/Book[book_ID=$c/book_ID/text()] else ()</condaction>
</rule>
```

**Figure 4.4 Crisp Rule Translation Output**
4.3 Fuzzy Terms in Database

The system built in [13] only allows crisp values in the database. This project allows specification of fuzzy terms in the database. Consider an example, the popularity in the Bookstore application can hold fuzzy values like “LOW”, “MED” or “HIGH” rather than just numerical precise values.

Values representing the different distributions discussed in Chapter 2 can be part of the fuzzy database. Examples of fuzzy values include <quantity_in_stock> LOW </quantity_in_stock> which maps to a trapezoidal distribution, <popularity> ~50 </popularity> which represents a triangular distribution and <popularity> [25, 40] </popularity> which relates to an interval distribution.

The min and max for the fuzzy valued attribute is manipulated using the same semantics as discussed for rules with fuzzy terms in section 4.2. A hidden table is generated where the fuzzy value has its numerical equivalents being calculated and stored. The need for a hidden table is to eliminate the performance overhead that might reduce efficiency of operation.

The DOM parser scans through the nodes in the database to identify the values. If a value is found to be fuzzy, its min and max values are calculated using the possibility distributions with a default threshold value of 1.0. For any attribute in the database, two nodes namely, attribute_min and attribute_max are appended to the table with its values. If the attribute is followed by a threshold value, it used to calculate min and max.
Consider an example: For a database entry - `<popularity> LOW </popularity>`,
`<threshold_value>0.8</threshold_value>`

The hidden table would have entries like,

`<popularity> LOW </popularity>`

`<threshold_value>0.8</threshold_value>`

`<popularity_min>40.0</popularity_min>`

`<popularity_max>110.0</popularity_max>`

4.4 Timer Rules

This project designed a timer rule language that allows users to specify events generated upon time intervals (*temporal events*). This project also designed and implemented the execution system that supports timer rule execution. A generic example of a timer rule would be “Check inventory at 10:00pm everyday”. Another example with respect to the Bookstore application would be, “If quantity in stock is low and popularity is high, and then insert Purchase Order nodes.

Any timer rule would follow the template as shown in Figure 4.5. A timer rule is uniquely identified by a rule name. A timer rule has four components to describe a rule. `<event_type>` specifies what type of event gets triggered by a rule (*temporal_event*). `<event_frequency>` specifies the delay in the timer rule execution. Possible values include: hourly, daily, weekly or monthly. `<start_time>` indicates the time for the rule to
be triggered for the first time and is specified in a 24 hour format. The `<condaction>` part specifies the condition and action to be satisfied for the timer rule to get triggered.

```xml
<rules>
  <rule>
    <rulename>Name of the temporal event</rulename>
    <event_type>temporal event</event_type>
    <event_frequency>freq of event</event_frequency>
    <start_time>start time of event</start_time>
    <condaction>condition and action</condaction>
  </rule>
</rules>
```

**Figure 4.5 Template for Timer Rule**

An example of a timer rule used to illustrate the semantics and implementation involved in processing timer rules to schedule new events is shown in Figure 4.6.

```xml
<rule>
  <rulename>Insert Discount Information</rulename>
  <event_type>temporal_event</event_type>
  <event_frequency>daily</event_frequency>
  <start_time>22:00</start_time>
  <condaction>
    let $a := collection('ss.dbxml')/Bookstore return
  </condaction>
</rule>
```

**Figure 4.6 An Example Timer Rule**
for $b in ($a/Book) return

    if($b/quantity_in_stock/text() > 2000 and
        $b/%popularity/text() < LOW with threshold 0.6) then
        if(exists($a/Discount_info/percent/text())) then ()
        else insert nodes <Discount_info>
            <percent>10</percent>
        </Discount_info> after $b else()

</condaction>
</rule>

**Figure 4.6 An Example Timer Rule**

The timer rule in Figure 4.6 holds the *event_type* to be *temporal_event*. The rule is triggered for the first time at 10:00pm and its frequency is 24 hours. The *condaction* part specifies that quantity_in_stock for a book is greater than 2000 and popularity is less than LOW, then insert discount of 10% to the book if one does not exist.

Figure 4.7 shows the how to drop timer rules which have been scheduled. Consider an example *drop Insert Discount Information*. This rule removes the timer with the rulename Insert Discount Information from the rule base.

**Figure 4.7 Template to Drop a Timer Rule**
4.4.1 Timer Rules Processing

Temporal events are entered by the user through the user interface and handled using the flow described in Figure 3.2. The new rule gets added to the timer rules repository (temporal_events.xml). The DOM parser is used to hierarchically trace down the xml tree structure and insert new events. The user enters a timer rule according to the template in Figure 4.5 to insert a new rule.

The Master cron job identifies that this is a new rule added to the system and it needs to be scheduled. The master job is a timer job scheduled to run in a given time interval (half an hour in this implementation) using the semantics as shown below:

```java
timer.scheduleAtFixedRate(instance.new set_temporal_event(), date, period);
```

set_temporal_event is the TimerTask in the implementation. The run method in TimerTask is the function acted upon at the expiration of every period. In this case the period is 30 minutes (30*60 seconds). The run function scans through the timer rules stored in the database and investigates the status node. If the status is `active`, no action is performed. If the status is `new`, event_freq, start_time and conduction along with rule name is read into temporary variables. A timer thread is invoked with its associated name to be the rule name of the event that is parsed. In the timer thread, period is calculated in terms of seconds for each of the accepted values, hourly, daily, weekly or monthly and the timer is scheduled as below:

```java
timer_object[thread_count].scheduleAtFixedRate(t1.new trigger_task(condaction), date, period);
```
The `trigger_task(codaction)` executes the actual condition and action part against the database. Once the timer is scheduled, the status of the timer rule is changed to “active”, indicating the event has been scheduled and is currently active in execution.

If the status of the rule was `cancel`, each timer threads associated name is queried. If the name matches the rule name passed as an argument, its associated timer is cancelled and the thread associated with it is interrupted. The query to remove rules with the passed in rule name from the temporal events repository is executed against the database. The code snippet showing the cancellation of a temporal event is shown in Figure 4.8.

```java
System.out.println("Number of scheduled timers:"+ thread_count);
if (thread_obj[i].getName().equals(rulename)) {
    System.out.println("Rulename:"+ thread_obj[i].getName()+" matches to cancel");
    timer_object[i].cancel();
    thread_obj[i].interrupt();
    System.out.println("Timer cancelled");
}
```

**Figure 4.8 Cancellation of Temporal Event**

The results of executing the timer rule in Figure 4.6 against the database are presented in Figure 4.9.
<Bookstore>

<Book>

  <book_ID>1</book_ID>

  <title>Harry Potter and the Order of the Phoenix</title>

  <author>
    <author_fname>J.K.</author_fname>
    <author_lname>Rowling</author_lname>
  </author>

  <price>9.99</price>
  <year_published>2004</year_published>

  <publisher>Scholastic, Inc.</publisher>
  <genre>Fiction</genre>

  <popularity>LOW</popularity>

  <quantity_in_stock>3200</quantity_in_stock>

</Book>

<Discount_info>
  <percent>10</percent>
</Discount_info>

Figure 4.9 Temporal Event Execution Result
Chapter 5

SYSTEM QUERYING AND PERFORMANCE ANALYSIS

In this chapter the basics involved in querying XML databases is presented. The performance analyses of executing crisp and fuzzy queries are also tabulated.

5.1 Database Query

XML databases are stored as persistent DOM. It enhances performance and drops the data right into the database without the need to manipulate or extract any information from the document. Querying against XML databases using XQuery provides the same breadth of functionality as SQL does for relational databases.

XQuery is a functional language where each query is an expression. It has the sophisticated type system based on XML schema datatypes and supports manipulation of the document nodes [15]. Based on XQuery 1.0 and the Bookstore application a set of sample queries have been created.

Figure 5.1 shows an example of a simple query where the book ids of all books with the quantity in stock less than HIGH are returned.

```
collection('dd.dbxml')/Bookstore/Book[quantity_in_stock/text() <= HIGH with
threshold 0.6]/book_ID
```

Figure 5.1 Simple Query
The query is composed using fuzzy term quantity\_in\_stock, denoted by \%quantity\_in\_stock. On identifying a fuzzy term, the instance of the class where fuzzy rules are converted to crisp rules is invoked, and the corresponding crisp rule is generated.

The results of executing the query in Figure 5.1 are as follows:

```
<book_ID>2</book_ID>
```

For more examples of fuzzy active queries and the converted crisp forms refer to Appendix D. The query results are also listed.

### 5.2 Performance Analysis

In software engineering, performance testing is the testing that is performed to determine how fast some aspect of a system performs under a particular workload [17]. In our XML database system, we analyze performance based on comparison of two different ways to represent XQuery that is executed with the database. One form of Xquery is the crisp one and the other is the fuzzy one, which needs to be converted to crisp and then executed.

The execution time of the model was calculated as a difference in time between the start and end of query execution. Figure 5.2 shows the code snippet that explains the calculation of execution time.
```java
Long startTime = System.currentTimeMillis();
query_execution();
long endTime = System.currentTimeMillis();
System.out.println("Total elapsed time in execution for a Crisp Query: \( \) + (endTime - startTime));
```

Figure 5.2 Query Execution Time

The Table 5.1 below shows the execution times of crisp and fuzzy rules in different scenarios. The Figures 5.3 and 5.4 shows a graphical representation of the query results for queries in Figure 5.1 and Appendix D. Series 1 is for crisp queries and Series 2 is for complex queries in the graphs. The values are subject to changes based on system load.

<table>
<thead>
<tr>
<th>Query execution scenario</th>
<th>Simple queries</th>
<th>Complex queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Crisp query</td>
<td>Fuzzy query</td>
</tr>
<tr>
<td>Queries run a single time</td>
<td>375</td>
<td>438</td>
</tr>
<tr>
<td>Queries run multiple times (10)</td>
<td>2630</td>
<td>3830</td>
</tr>
</tbody>
</table>

Table 5.1 Performance Analyses Report
From the graphs, it is clearly understood that the average overhead involved in fuzzy active queries translation is only 10-20% when compared to crisp queries.
Software testing is an activity aimed to evaluate an attribute of a program or a system to determine if its meets the requirements [17]. This chapter gives a brief idea of the different tests which were performed to verify the correctness of generating crisp rules from fuzzy rules, querying against a database which has fuzzy values, and monitoring performance of temporal events.

6.1 Validation of Fuzzy Parser

The fuzzy parser is the main module involved in the translation of fuzzy active rules to crisp rules. Testing was done on an incremental basis to ensure correctness of functionality. The first step was to make sure parser reported an error for invalid characters. The only symbol accepted to represent a term as fuzzy is %. If any other symbol was made part of the fuzzy rule, a parser error would be generated as shown in Figure 6.1.

```java
Exception in thread "main" FuzzyDB.TokenMgrError: Lexical error at line 1, column 366. Encountered: "" (96), after : "" at FuzzyDB.Fuzzy_Rule_ParserManager.nextToken(Fuzzy_Rule_ParserTokenManager.java:783)
at FuzzyDB.Fuzzy_Rule_Parser.jj_ntk(Fuzzy_Rule_Parser.java:2158)
```

Figure 6.1 Exception on an Invalid Character to Represent Fuzzy Term
The completeness required in Fuzzy parser was also tested. For example, the threshold value should be specified for fuzzy terms. In cases where it fails, an exception should be generated as shown in the Figure 6.2.

```
Exception in thread "main" FuzzyDB.ParseException: Encountered " <Letter> "a "" at line 1, column 392.
Was expecting:
"with" ... at
FuzzyDB.Fuzzy_Rule_Parser.generateParseException(Fuzzy_Rule_Parser.java:2198)
at FuzzyDB.Fuzzy_Rule_Parser.jj_consume_token(Fuzzy_Rule_Parser.java:2133)
at FuzzyDB.Fuzzy_Rule_Parser.FuzzyComparison(Fuzzy_Rule_Parser.java:828)
at FuzzyDB.Fuzzy_Rule_Parser.event(Fuzzy_Rule_Parser.java:575)
at FuzzyDB.Fuzzy_Rule_Parser.main(Fuzzy_Rule_Parser.java:58)
```

**Figure 6.2 Exception for Missing Term in a Rule**

The DOM parser in our application traces through the XML tree in a hierarchical fashion. If a particular node is missing in a rule, an exception should be generated. It is basically to notify the user that a required value was not provided.

```
Event is: Insert PuException in thread "main" java.util.NoSuchElementException
at java.util.Scanner.throwFor(Unknown Source)
at java.util.Scanner.next(Unknown Source)
at FuzzyDB.Fuzzy_Rule_Parser.main(Fuzzy_Rule_Parser.java:198)
```

**Figure 6.3 Error for Missing Node**
6.2 Validation of Crisp Rule

The fuzzy active rules given as input are translated into the crisp rule by following the flow shown in Figure 3.1. The verification of the generated crisp rule was done by running the condition action part against the database using Berkeley XML command line option to see if the intended results were achieved. Further validation of the generated crisp rule is by using it for temporal events automation and is explained in section 6.4.

6.3 Validation of Querying

Querying against XML database were done in two parts. The first was to cover the correctness of querying and the results generated using a database without any fuzzy values. The next was to include fuzzy terms in the database and run queries against it.

Queries were run on both the command line of Berkeley XML DB and also through code as described in section 5.1. The range of queries tested include very simple queries which return a single value, to complex queries which do a fuzzy term translation to get to a precise note.

Update queries were run against the database and the reflected values were cross verified to ensure correctness. The results of running queries from the condition action part should insert new nodes under a particular book. The placement of nodes was verified to maintain tree balance. An example of running the Insert Discount Information rule should insert nodes as shown in Figure 6.4.
<Bookstore>

<Book>

  <book_ID>1</book_ID>

  <title>Harry Potter and the Order of the Phoenix</title>

  <author>
    <author_fname>J.K.</author_fname>
    <author_lname>Rowling</author_lname>
  </author>

  <price>9.99</price>

  <year_published>2004</year_published>

  <publisher>Scholastic, Inc.</publisher>

  <genre>Fiction</genre>

  <popularity>LOW</popularity>

  <quantity_in_stock>3200</quantity_in_stock>

</Book>

<Discount_info>
  <percent>10</percent>
</Discount_info>

Figure 6.4 Testing Insertion of Nodes in DB
Correctness of running delete queries was also performed. When fuzzy terms were included in the database, the hidden table was tested to include new nodes indicating the min and max value calculated with a threshold value of 1.0. Queries run against the database with fuzzy terms should get translated to indicate either the min or max value shown in Figure 6.5. The combinations of query comparisons in condition part were also tested to verify correctness in degree of satisfaction that was calculated.

```
$b/((\text{quantity\_in\_stock} \geq 1190.0) \text{ and } (\text{quantity\_in\_stock} \leq 1240.0)) \text{ and } ((\text{popularity\_min} \geq 560.0) \text{ and } (\text{popularity\_min} \leq 860.0)) \text{ or } ((\text{popularity\_min} \leq 560.0) \text{ and } (\text{popularity\_max} \geq 860.0)) \text{ or } ((\text{popularity\_min} < 560.0) \text{ and } (\text{popularity\_max} < 860.0)))
```

**Figure 6.5 Correctness of Database Fuzzy Term Interpretation**

### 6.4 Validation of Temporal Events

The verification of operating correctness for temporal events was limited to adding new events to schedule new timers or drop existing timers. The master cron job was tested only to a certain extent as one JVM could not handle parallel execution of two java programs. When new rules were added to the temporal events repository, the status node was verified to hold the correct value as it drives the entire flow in our implementation.

The main cron job was run on a day basis to make sure the timer gets activated every
half an hour and scans through the list of events in the temporal events database. The rules that were added to schedule new rules got triggered as the status had a value ‘new’. Its corresponding timer thread was activated with the rule name assigned as a unique parameter to identify a thread. The events with the status value ‘active’ were monitored to ensure no action was performed. For cancellation the testing was limited to adding the cancel event to the database. For activation of cancel rules, the timer should have been scheduled and main cron job should pick up the rule at its trigger period. The correctness was however tested by running cancel events and monitoring the temporal events database to have removed two events with the corresponding rule name.
Chapter 7

SUMMARY AND FUTURE WORK

The project implemented the different distributions that can be made part of the fuzzy active rules, storage of fuzzy terms in the database and language definition and automization of timer rules. The language was created with an involvement to make sure the user was not overloaded to remember too many symbols to represent fuzzy or crisp terms. Neither were too many nodes were included in the template for adding a temporal event to the database.

The performance analysis done for comparing crisp and fuzzy query rules shows the overhead for processing fuzzy queries. The overhead involved in translation is ignorable when compared to the benefits the user would experience on specification of rules and events that are automated.

The future work would include enhancements to user interface for the entire application. A much more refined version of data set can include a wide range of events and rules to be part of the system. The data set used currently, restricts the error handling functionality. Most of the exception handling in the current implementation is limited to query results and corresponding changes to the database. A better automated testing mechanism might catch a few more corner cases. Running the main cron job on a server would give us an opportunity to realistically calculate the time consumed and overhead involved in handling temporal events.
APPENDIX A

Bookstore Application

This section provides the details on the tree structure of the Bookstore database, based on the bookstore database in [13]. The element at the top level is named Bookstore, and it contains five child elements, each of which in turn contains one or more child elements, as shown below.

**Bookstore Database**

```
Bookstore
  ↓  ↓  ↓  ↓  ↓
BookSales Record Purchase order Book Sales History Discount Information
  ....  ....  ....  ....  ....
```
1. **Book Node**

```
Book --> Book_ID
      --> Title
      --> Subtitle
      --> Author
      |   --> Author Fname
      |   --> Author Lname
      --> Price
      --> Year_Published
      --> Publisher
      --> Genre
      --> Popularity
      --> Quantity_In_Stock
```

2. **Sales Record Node**

```
Sales Record --> Book_ID
               --> Sale_Date
               --> Quantity_Sold
               --> Sale_Price
```
3. **Purchase Order Node**

   ![Purchase Order Diagram]

4. **Discount Information Node**

   ![Discount Information Diagram]
5. Book Sales History Node

Book Sales History

Book_ID

Year

ID

- January
- February
- March
- April
- May
- June
- July
- August
- September
- October
- November
- December
APPENDIX B

Valid Tokens

This section provides a list of Valid Tokens accepted by the implementation module.

/*Lexical Analyzer*/

SKIP :
{
  "\r"
  | "\t"
  | "\n"
}

TOKEN : /* OPERATORS */
{
  < PLUS: "+" >
  | < MINUS: "-" >
  | < MULTIPLY: "*" >
  | < DIVIDE: "div" >
}

TOKEN:
{
  < Equals_Operator : "=" >
APPENDIX C

Fuzzy Parser

This section provides a layout of fuzzy parser as per the semantics involved in this project.

```java
{  
  (t = <PERCENT>
  )
  (str = LinVar()
    {
      linguistic_variable = str;
      String term = GetTerm(linguistic_variable);
      System.out.println("***Term:" +term);
      if(term.equals("fuzzy")) {fuzzy_term = true;}
    }
  )
  (t = <Slash>
    {
      linguistic_variable_rhs += t.image;
    }
  )*

  (short_string = ValidCharacter()
    {
      linguistic_variable_rhs += short_string;
      System.out.println("linguistic_variable_rhs in Fuzzy Comp :" +linguistic_variable_rhs);
    }
  )+
  (t = <space>
  )*
  // Parse the operator /quantity_in_stock/text() < LOW ;
  t = Operator()
  }
```
comparison_operator = t.image;
    System.out.println("Comparison_op:" + comparison_operator);
}
(t = <space>}
    { }
}*)

// Parse for any approximate op
(t = <Approx>
{ approx_hit = true;
    System.out.println("Approx symbol hit");
    input_value = GetValue();
    System.out.println("Value returned from GetValue() +input_value");
}
(t = <space>}
    { thresholds += t.image;
        System.out.println("thresholds in Fuzzy Comp:" + thresholds);
    }
(t = <with>
{ thresholds += t.image;
        System.out.println("thresholds in Fuzzy Comp:" + thresholds);
    }
(t = <space>}
    { thresholds += t.image;
        System.out.println("thresholds in Fuzzy Comp:" + thresholds);
    }
(t = <threshold>
{ thresholds += t.image;
        System.out.println("thresholds in Fuzzy Comp:" + thresholds);
    }
(t = <space>}
    { thresholds += t.image;
        System.out.println("thresholds in Fuzzy Comp:" + thresholds);
    }
(t = <Threshold_Value>
{ thresholds += t.image;
    threshold_value = Double.parseDouble(t.image);    // translate from String into double
// Parse for any [ op to determine if an interval is specified
(t = <OpenSqBrace>
{
    System.out.println("Interval hit");
    interval_hit = true;
    System.out.println("[ symbol hit");
    interval_min = GetValue();
}
(t = <comma>
{
    interval_max = GetValue();
    System.out.println("Max value from GetValue() +interval_max");
}
(t = <CloseSqBrace>

(t = <space>
{
    thresholds += t.image;
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
(t = <with>
{
    thresholds += t.image;
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
(t = <space>
{
    thresholds += t.image;
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
)

(t = <threshold>
{
    thresholds += t.image;
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
(t = <space>
{
    thresholds += t.image;
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
t = <Threshold_Value>
{ thresholds += t.image;
    threshold_value = Double.parseDouble(t.image);   // translate from String into double
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
)*

// Parse the Linguistic Term like LOW, HIGH
str = LinguisticTerm(linguistic_variable)
{ linguistic_term = str;
}

(t = <space>
{ thresholds += t.image;
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
( t = <with>
{ thresholds += t.image;
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
) )

(t = <threshold>
{ thresholds += t.image;
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
(t = <space>
{ thresholds += t.image;
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
)

(t = <Threshold_Value>
{ thresholds += t.image;
    threshold_value = Double.parseDouble(t.image);   // translate from String into double
    System.out.println("thresholds in Fuzzy Comp:" +thresholds);
}
)*)
APPENDIX D

Fuzzy Active Queries

Fuzzy query:

let $a := \text{collection('dd.dbxml')/Bookstore return}$

for $b$ in ($a$/Book) return

if($b/%quantity_in_stock/text() < \sim 1200$ with threshold 0.6 and

$b/%popularity/text() > \sim 50$ with threshold 0.8) then $b/book\_ID$ else ()

Converted crisp query:

let $a := \text{collection('dd.dbxml')/Bookstore return}$

for $b$ in ($a$/Book) return

if($b/quantity\_in\_stock/text() < 1180.0$ and

$b/popularity\_max/text() > 54.0$) then $b/book\_ID$ else ()

Query results:

**********************************************

Query results:

Number of tuples: 1

<book\_ID> 2 </book\_ID>

**********************************************
REFERENCES

1. E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," CACM 13, No. 6, June 1970


11. E. N. Hanson, L. X. Noronha, “Timer-Driven Database Triggers and Alerters: Semantics and a challenge”, SIGMOD Record, vo. 28, No. 4, December 1999


