EPiC Series in Computing

Volume 82, 2022, Pages 82–91

Proceedings of 37th International Conference on Computers and Their Applications

Event and Query Processing in a Fuzzy Active Graph Database System

Ayushi Vadwala and Ying Jin
Department of Computer Science,
California State University, Sacramento,
Sacramento, CA 95819-6021, USA
ayushivadwala@gmail.com, jiny@csus.edu

Abstract

There is a lot of uncertain and imprecise information in a real-world scenario. Graph database systems based on the graph model support crisp and precise data. This paper presents a system structure that processes fuzzy quantified queries in the context of a graph database. Fuzzy logic allows decisions to be taken more realistically. A specific form of structural quantified query is designed and demonstrated in this paper, which then can be expressed as an extension to the Neo4j Cypher query language. In addition, this research built an active rule system on top of the graph database that reacts to event occurrences automatically. This paper presents our approach of supporting temporal event handling using fuzzy active rules and fuzzy query processing over Neo4j graph database systems.

1 Introduction

The amount of data is increasing both in terms of scale and connectivity, where connectivity refers to the relationships present among them. The processing of data became more onerous. Graph databases like neo4j [1] are developed to solve these issues. Graph databases are optimized and highly connected. It is mainly used for the data whose focus is on the relationship between the entities. The three main advantages of graph databases are performance, flexibility, and agility.

Generally, crisp and precise data are required by databases. However, when querying databases, users may want to convey vague ideas, such as searching for "cheap" hotels. Fuzzy logic overcomes the limitations of rigid crisp logic boundaries, allowing a way to take decisions realistically.

Passive database applications only perform specifically submitted queries or transactions by a user or application software. An active database system, however, is a database system that tracks situations of interest and, when they arise, triggers a suitable response on time. The required response of such events is expressed in condition and action parts of rules. This type of system gives users control over the process.

This paper describes an extension to the neo4j graph database to include the support of fuzzy queries and active fuzzy rules. We focus on timer rules that are rules triggered by time or time intervals. Compared to the related research that only incorporates fuzzy logic into graph databases, presented in Section 2.4, our system enhances graph databases by both fuzzy logic and active rules. Specifically, our system includes a language model to define fuzzy active rules and fuzzy queries, as well as an execution model to execute the rules upon event occurrence. In addition, our system supports fuzzy rules defined using different linguistic variables. Our system is generic that can be used for any application, however, to illustrate the concepts clearly, we use an example of a supply chain system with a list of retailers, distributors, products, and customers in this paper. The rest of the paper is organized as follows. Section 2 provides information on the background of the tools and technologies used. Section 3 presents the language model. Section 4 describes system design and rule execution flows. Section 5 describes the details of implementation. Section 6 presents the evaluation of the system's performance. Section 7 is the summary.

2 Related Work

2.1 Fuzzy Concepts Overview

In 1965, Lotfi A. Zadeh published "Fuzzy Sets", which introduced the concepts of fuzzy sets and then it evolved into fuzzy logic [2]. Fuzziness occurs when the boundary of a piece of information is not precise. For example, words such as young, tall, good, or high are fuzzy. Human reasoning often involves fuzzy information. Fuzzy logic extends computer capabilities to deal with imprecise and vague information. Fuzzy set theory is a generalization of the crisp sets and methods to resolve the different forms of information representation [2]. "The membership function in a fuzzy set is not a matter of true or false but a matter of degree" [2]. For example, "He is running fast", here "running" is a linguistic variable or fuzzy variable and "fast" is a linguistic value. The linguistic values can be converted to numeric values based on membership functions. Different distributions may use different membership functions.

2.2 Representation of Knowledge

There may be different representations of the elements relating to fuzzy data handling. Every linguistic variable can have a set of linguistic values that describe them, these sets can be retrieved from Fuzzy Metaknowledge Base (FMB). The FMB contains all the necessary information about the imprecise data the user can input [3]. There are majorly three types of knowledge representation [4]:

- Trapezoidal Distribution for Linguistic label

In this method, the linguistic variables are represented by a trapezoidal distribution. Each linguistic variable is associated with α, β, γ, and δ values. Based on α, β, γ, and δ, we can convert fuzzy values into crisp values. In addition, a threshold can be used in any distribution, which is a value between 0 and 1. If the threshold is not provided then the default value of the threshold is taken as 0. Based on the threshold value, the min and max are calculated using the following formulae [4]:

$$Min = (\beta - \alpha) * THOLD + \alpha$$

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

- Possibility Interval for Linguistic label

The linguistic variable is defined as an interval [m, n]. In possibility interval "m" and "n" denote the range of the linguistic term. Here "m" denotes the min value and "n" denotes the max value.

- Triangular Distribution for Linguistic label

In this method, the linguistic variables are represented by a triangular distribution. The linguistic variable is associated with a vague concept like "approximately d". "d" is the "median" value. Margin is the allowed maximum deviation from the median value. Triangular distribution enables the membership function to calculate the numerical value with a threshold between 0 and 1. Based on the threshold value the formula shown below calculates the min and max values:

Min = d – (margin (1 - THRES))*

*Max = d + (margin * (1 - THRES))*

2.3 Tools and Technology

Graph databases are a category of NoSQL databases that provide an effective and efficient solution for the storage of information in the current context, where data is very strongly interlinked. We are using Neo4j [1], a popular graph database, in this project.

Neo4j mainly has two basic building blocks – nodes and edges. Cypher Query Language also is known as CQL, is a query language for Neo4j. Cypher is a declarative pattern-matching language. We can delete, insert, and create basic elements such as nodes, edges, and properties using this query language.

2.4 Related Research Projects

Some research worked in the field of deriving crisp queries from fuzzy queries. Pivert, O. et al. discussed in [5] about the formulation of syntax and semantics of an extension of the Cypher query language that makes it possible to express and interpret fuzzy queries. The results of their experiments show that the cost of converting a fuzzy query to a crisp query is reasonable, considering the cost of the overall evaluation. Costa, B. P., and Cura, L. D. V. deliberated the use of fuzzy data in Graph Databases in [6]. They extended the work of fuzzy queries on the graph database by allowing users to store information in the fuzzy and imperfect data in the database itself. The membership function used is the trapezoidal distribution. Pivert, O. et.al in their paper [7] described a framework to introduce fuzzy terms into graph database queries. The framework used the Neo4j engine, and combined it with an add-on dedicated layer that carries out pre-processing and postprocessing of the fuzzy quality queries. The add-on layer consists of a compiling module that transforms the queries with fuzzy quality into crisp Neo4j Cypher queries, as well as a quality score calculation module. Castelltort, A., & Martin, T., in their article [8] introduced an approach to define and run an approximate queried on a graph database. The proposed approach uses Scala to propose the Fuzzy4S framework and Cypherf fuzzy declarative query language. The membership function used by the author is the triangular distribution for the linguistic label.

Compared to the above research, we implement all of the three membership functions in fuzzy queries, so that users have the flexibility to provide the type of fuzzy input. In addition, our system

incorporates fuzzy active rules into the graph database, so that corresponding actions will be executed if a specific event occurs. Our previous research worked on incorporating crisp active rules into graph databases [9] and introducing fuzzy logic into XML databases [10], [11].

3 Language Model

In this paper, we use a supply chain application to illustrate fuzzy rules and fuzzy queries. The supply chain database is a graph database that consists of a series of nodes related to retailers, suppliers, customers, and products. The linguistic variables in the Supply chain application include *productPopularity, inventoryAvailabilityPercent,* and *profit.* For each item, a store has an inventory capacity, which is the max number of items that can be possibly in stock. Item quantity decreases when an item is sold; quantity increases when a new shipment arrives. The *inventoryAvailabilityPercent* is the ratio of current quantity divided by inventory capacity. When this ratio is low, the store may want to reorder the item from a supplier.

Figure 1 presents the linguistic variable "inventoryAvailabilityPercent" in the Fuzzy Metaknowledge Base (FMB). Other linguistic variables are also stored in FMB, with their corresponding linguistic terms and values.

```
"inventoryAvailabilityPercent": {
        "margin": "20",
        "high": [
                {"alpha": "70"},
                {"beta": "75"},
                {"gama": "95"},
                {"delta": "100"}],
        "low": [
                {"alpha": "10"},
                {"beta": "15"},
                {"gama": "25"},
                {"delta": "30"}],
        "moderate": [
                {"alpha": "40"},
                {"beta": "45"},
                {"gama": "55"},
                {"delta": "60"} ]}
```

Figure 1: Linguistic Variables and Values in the FMB

Based on the definitions in FMB, a crisp value can be calculated based on its corresponding possibility distribution, such as trapezoidal. For example, the calculation for Trapezoidal Distribution for Linguistic label can be done as follows for "*inventoryAvailabilityPercent = #High WITH THOLD = 0.5*":

First, applying formulas for Trapezoidal:

$Min = (\beta - \alpha) * THOLD + \alpha = (75\text{-}70) * 0.5 + 70 = 72.5$

$Max = [(\delta - \gamma) * (1 - THOLD)] + \gamma = [(100\text{-}95) * (1\text{-}0.5)] + 95 = 97.5$

Next, we can get min value of 72.5 and max value of 97.5. The interval for availability percent can be written as "*inventoryAvailabilityPercent >= 72.5 and inventoryAvailabilityPercent <= 97.5*". Once linguistic variables and values are defined in FMB, users can express a fuzzy query using fuzzy values in lieu of crisp values.

Compared to a crisp Cypher query, a fuzzy query consists of linguistic variables and linguistic terms. Within a fuzzy query, the following special characters are used to identify different distributions:

- The special character '#' identifies the use of trapezoidal distribution for the linguistic label
- The special character '[]' identifies the use of interval distribution for the linguistic label.
- The special character '~' identifies the use of triangular distribution for the linguistic label.
- The linguistic variable is followed by one of the relational operators (<, <=, >, >=, =, !=).

An example of a fuzzy query is shown in Figure 2. It presents the following business logic: in the supply chain application, we want to go through all the products and check the inventory availability percent, and then return product name and other specified attributes which satisfy the approximate value condition.

In addition to fuzzy queries, our system supports fuzzy active rules. An active rule consists of an event, a condition, and an action. A fuzzy active rule allows users to use fuzzy queries within the condition and action of the rule. An active rule can be triggered by mutation events, such as create, insert, and delete operations, for example, when a product's quantity decreases. Another type of event is a temporal event. A temporal event happens upon a time instance. Rules triggered by temporal events are called timer rules. We only cover the timer rule in this paper.

```
MATCH (n:CostRetailerProduct)
WHERE n.inventoryAvailabilityPercent = ~30 WITH THOLD = 0.5
RETURN n.name AS name, n.productPopularity AS productPopularity, n.discount AS
discount, n.cost AS cost, n.restock as restock, n.inventoryAvailabilityPercent AS
inventoryAvailabilityPercent
```

Figure 2: Fuzzy Query Example

A fuzzy active rule is uniquely identified by a rule name. A fuzzy active rule consists of three major components: Rule name, Event, and Condition and action. "Rule Name" specifies the unique name of the rule. "Event" has two subcategories: "StartTime" and "Frequency". "StartTime" gives the time at which the rule is to be scheduled for its initial run. The input format of the time should be HH:MM: SS aa. "Frequency" specifies how often the rule is triggered, for example, every hour. The input format of the frequency is in hours. The "Condition and Action" part contains a fuzzy query or crisp query and covers the main business logic of the rule.

Figure 3 shows an example of a fuzzy active rule that specifies discounts for an item depending on profit, popularity, and inventory. Every day at 8 pm, if the product's profit is between 30 and 40 percent, the product popularity is low with the threshold of 0.5, and the inventory availability is approximately 80 percent with the threshold of 0.4, then the discount for that product will be set to 10% of the original price. At the scheduled time i.e. 08:00:00 pm, with the frequency of every 24 hours, this rule will be executed. In the next section, we will present how to process rules and fuzzy queries in our system.

```
{    "Rule": [
        {"Rule Name": "Discount"},
        {"Event": [
                {"StartTime": "08:00:00 pm"},
                {"Frequency": "24"}]},
        {"Condition and Action": MATCH (c:CostRetailerProduct) WHERE c.profit = [30,40]
AND c.productPopularity = #low WITH THOLD = 0.5 AND c.inventoryAvailabilityPercent =
~80 WITH THOLD = 0.4 SET c.discount = toString("10%")}] }
```

Figure 3: An Example of Fuzzy Active Rule

4 System Design

This section describes the architecture and design of the execution of timer rules and fuzzy queries.

4.1 Timer Rules Handling

The architecture to handle the timer rules is shown in Figure 4. The system includes major components such as user interface, fuzzy and crisp rule repository, neo4j graph database, and fuzzy parser. The system contains two repositories that are used at different stages of the execution. They store the rules in two different formats respectively: fuzzy and crisp. A user can create a new rule, delete, or view an existing rule. To delete or view an existing rule a user needs to provide the rule name. When creating a new rule, a user inputs start time, frequency, and condition, and action. The newly created rule is stored in the Fuzzy Rule Repository. Next, start time, frequency, and condition, and action are extracted from the rule in the Fuzzy Rule Repository. The condition and action part of the rule is passed to the Fuzzy Parser.

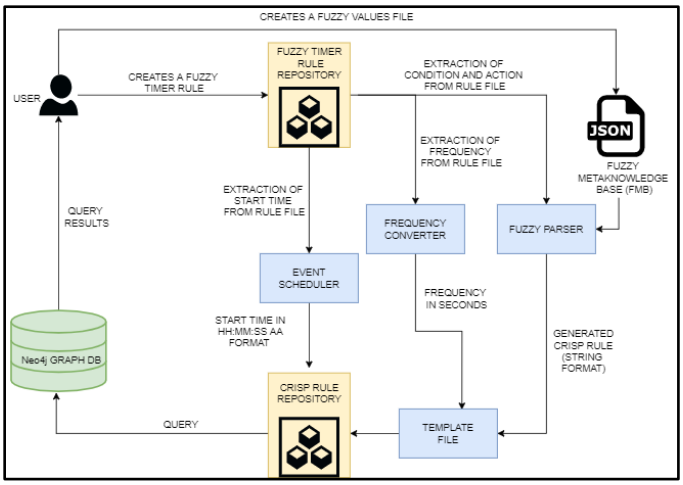

Figure 4: System architecture of timer rules handling

Using values stored in the Fuzzy Metaknowledge Base (FMB), the fuzzy parser converts a fuzzy rule into a crisp rule. We will discuss the details about how to convert in the next section. The converted crisp rule is stored in the Crisp Rule Repository. The Event Scheduler schedules the rule stored in the Crisp Rule Repository using the start time given by the user initially. At the scheduled time, at the

frequency specified, the query is sent to the Neo4j graph database for execution. The results provided by the database are then outputted to the user.

4.2 Fuzzy Query Handling

The flow diagram of converting a fuzzy query to a crisp format is shown in Figure 5. It includes major components like user interface, fuzzy and crisp queries, Neo4j graph database, and fuzzy parser.

A user interacts with the system with a newly created fuzzy query. The query is forwarded to the Fuzzy Parser. The Fuzzy Parser converts the fuzzy query to a crisp query based on the definitions in FMB. The generated crisp values are passed to the Neo4j graph database. The output, given by Neo4j, is sent back to the user. We will discuss the details of how to convert a fuzzy query to a crisp query in the next section.

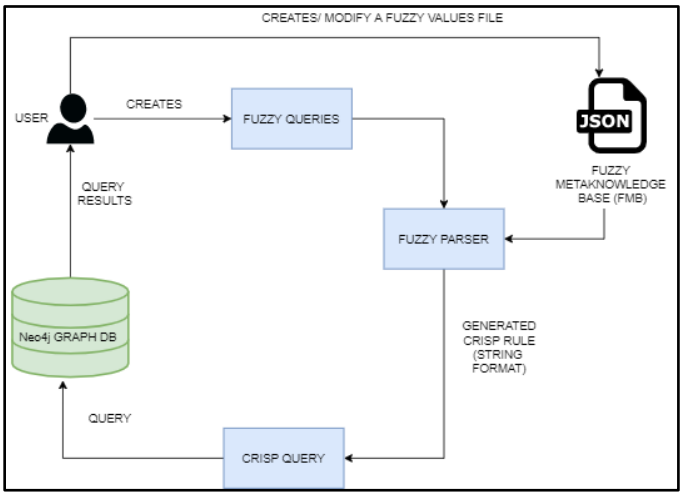

Figure 5: System architecture for fuzzy query handling

5 System implementation

This section describes the implementation details of processing active rules and handling fuzzy queries.

5.1 Transformation from Fuzzy to Crisp

A fuzzy parser is a form of a string analyzer that processes from left to right on the fuzzy query and identifies fuzzy terms based on the language. The lexical analyzer in the parser breaks the input into a series of tokens and then identifies the type of the token.

The Fuzzy Parser scans character-based inputs. If the character matches one of the tokens in the valid character set, the parser will continue to loop. If not, a parser error will be created with the "Invalid Token" message. The parser loops in a routine which identifies the linguistic variable using FMB and then checks if the rest of the terms satisfies the specific format.

The Fuzzy Parser identifies the fuzzy variable and then scans to understand the relational operator in the expression. If the operator is followed by '#' then it searches for the linguistic term after that and the values of α, β, γ, and δ are obtained from the FMB under the stored linguistic variable and linguistic term. If the operator is followed by a "~", the value of margin under the specified linguistic variable is chosen. If the parser comes across "[" then it handles the distribution as interval distribution. The parser then continues scanning and obtains the threshold values. 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.

Using the aforementioned translation parameters, the crisp rule is generated from the fuzzy query input. The condition and action part containing fuzzy terms is converted to the form of crisp values. For example, the fuzzy rule in Figure 3 is converted to a crisp rule in Figure 6.

```
{     "Rule": [
        {"Rule Name": "Discount"},
        {"Event": [
                {"StartTime": "08:00:00 pm"},
                {"Frequency": "24"} ]},
        {"Condition and Action": "MATCH (c:CostRetailerProduct) WHERE c.profit  >= 30.0
AND c.profit <= 40.0 AND c.productPopularity  >= 1.25 AND c.productPopularity <= 2.75
AND c.inventoryAvailabilityPercent  >= 68.0 AND c.inventoryAvailabilityPercent <= 92.0
SET c.discount = toString("10%")"} ] }
```

Figure 6: Translated Crisp Rule

5.2 Timer Rule Processing

After a fuzzy active rule stored in the database, upon event occurrence, the rule will be executed automatically. To make the rule executable over Neo4j, we have formatted the rules. The format contains the rule name, condition and action, and the frequency. These arguments can be extracted from the rule file in Figure 6. The frequency is converted into seconds. The active rule is eventually transformed into an *apoc* call to the neo4j graph database. Figure 7 is an example of final transformation of the rule in Figure 6.

The initial scheduling of the rule is done by the Java scheduling libraries, and the subsequent periodic scheduling is handled by Neo4j. For example, a rule is created at 5 PM on a day and the scheduled time is 8 PM with an interval of 24 hours. The first-time scheduling of the rule will be done by Java scheduling libraries at 8 PM and the subsequent scheduling of the rule will be executed by Neo4j every day at 8 PM.

```
call apoc.periodic.repeat('Discount', 'MATCH (c:CostRetailerProduct) WHERE c.profit  >=
30.0 AND c.profit <= 40.0 AND c.productPopularity  >= 1.25 AND c.productPopularity <=
2.75 AND c.inventoryAvailabilityPercent  >= 68.0 AND c.inventoryAvailabilityPercent <=
92.0 SET c.discount = toString("10%")', 86400)
```

Figure 7: Final Query Format

5.3 Fuzzy Query Processing

Instead of using Cypher to specify a query with crisp values, users can use fuzzy query to retrieve data from Neo4j. A fuzzy query will be converted into a crisp query firstly. Recall that the condition

and action of an active rule can be a fuzzy query. Similar to converting condition and action of a rule, a crisp query is produced by the process described in Section V(A). A converted crisp query will be sent to Neo4j and the execution results will be returned to the user. For example, the fuzzy query shown in Figure 2 is converted to a crisp query shown in Figure 8.

```
MATCH (n:CostRetailerProduct)
WHERE n.inventoryAvailabilityPercent  >= 20.0 AND n.inventoryAvailabilityPercent <= 40.0
RETURN n.name AS name, n.productPopularity AS productPopularity, n.discount AS discount,
n.cost AS cost, n.restock as restock, n.inventoryAvailabilityPercent AS
inventoryAvailabilityPercent
```

Figure 8: Converted Crisp Query

6 System performance analysis

The performance was evaluated in the environment of 8GB RAM and Intel i5 10th Gen processor. In our Neo4j graph database system, we examine the performance based on a comparison of two different ways to represent Cypher that is executed within the database. The first form of the query is fuzzy and the second one is crisp. The execution time of the model is calculated as a difference in time between the start and end of query execution. In table 1 three types of queries are used queries with one, two, and three linguistic variables. Figure 9 shows the bar graph comparison for queries with the average value when runs 100 times.

Query with 1 Linguistic Variable		**Query with 2 Linguistic Variable**		**Query with 3 Linguistic Variable**	
Crisp query	***Fuzzy query***	***Crisp query***	***Fuzzy query***	***Crisp query***	***Fuzzy query***
1142.79 seconds	1143.64 seconds	1145.42 seconds	1147.59 seconds	1173.89 seconds	1177.31 seconds

Table 1: Performance analysis report

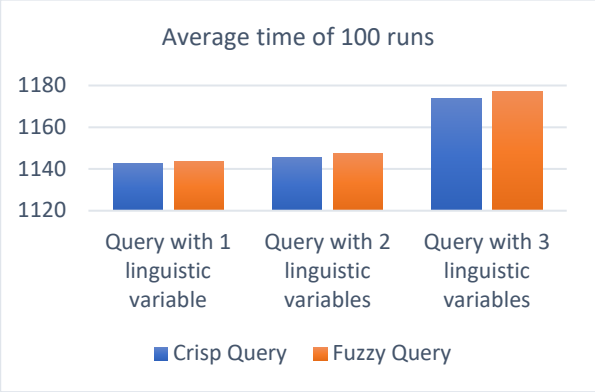

Figure 9: Average time of 100 runs

7 Conclusion

In traditional database systems, crisp and precise data are required. However, users may also wish to convey vague concepts when querying databases. It can be specified by using linguistic terms in lieu of crisp values. This paper presents different distributions such as trapezoidal distribution, possibility interval, and triangular distribution to represent linguistic terms in the query. These distributions are used in fuzzy active rules and fuzzy queries. Our fuzzy query definition is based on Neo4j's Cypher Query Language, where the user can specify the values of the linguistic variable as a linguistic term. The efficiency analysis for the crisp and fuzzy query rules presents that the overhead for the processing of fuzzy queries is small in comparison with the queries with crisp values.

The main contribution of this research is that this is the first project that incorporates fuzzy active rules into a graph database system, to the best of our knowledge. Our future direction is to incorporate fuzzy terms into active rules that triggered by mutation events.

References

[1] Neo4j Team, "Neo4j Documentation".[Online] Available: https://neo4j.com/docs/.

[2] Goguen, J. A. (1973). LA Zadeh. "Fuzzy sets," *Information and control*, vol. 8 (1965), pp. 338–353.-LA Zadeh. Similarity relations and fuzzy orderings. Information sciences, vol. 3 (1971), pp. 177–200. The Journal of Symbolic Logic, 38(4), 656-657.

[3] J. M. Medina, M. A. Vila, J. C. Cubero, and O. Pons, (1995). "Towards the implementation of a generalized fuzzy relational database model," *Fuzzy Sets and Systems*, 75(3), 273-289.

[4] S. Škrbić, M. Racković, and A. Takači, "Prioritized fuzzy logic based information processing in relational databases," *Knowledge-Based Systems*, ,2013, 38, 62-73.

[5] O. Pivert, O. Slama, and V. Thion, "Expression and efficient evaluation of fuzzy quantified structural queries to fuzzy graph databases," *Fuzzy Sets and Systems*, 2019, 366, 3-17.

[6] B.P. Costa, and L. D. V. Cura, "An Neo4j implementation for designing fuzzy graph databases," in *Proceedings of the 23rd International Database Applications & Engineering Symposium* , June 2019, pp. 1-6.

[7] O. Pivert, E. Scholly, G. Smits, and V. Thion, "Fuzzy quality-Aware queries to graph databases," *Information Sciences*, 2020, 521, pp. 160-173.

[8] A. Castelltort and T. Martin "Handling scalable approximate queries over NoSQL graph databases: Cypherf and the Fuzzy4S framework," *Fuzzy Sets and Systems*, 2018, 348, 21-49.

[9] Y. Jin, V. L. V. S. R. Bharath and J. Shah, "Active Rule in a Graph Database Environment," in p*roceedings of 35th International Conference on Computers and Their Applications*, March 23-25, 2020, San Francisco, California, USA, pp. 134-140

[10] Y. Jin, H. J. Mehta, and C. Madalli, "An Active Rule-based Fuzzy XML Database System," *Journal of Computational Methods in Science and Engineering (JCMSE)* 12 (2012), IOS Press, The Netherlands, pp. 103-111

[11] Y. Jin, and H. J. Mehta, "Composite Event Processing in an Active Rule-Based Fuzzy XML Database System," in *proceedings of the 12th IEEE International Conference on Information Reuse and Integration*, August 3-5, 2011, Las Vegas, NV, USA, pp. 7-10