Symposium on Tool Environments and Development Methods for Intelligent Systems (TOOLMET2000), April 13-14, 2000, Oulu, Finland.

Quantified and Temporal Fuzzy Reasoning for Active Monitoring in RapidBase

Antti Pesonen
Antoni Wolski

VTT Information Technology
P.O. Box 1201, 02044 VTT, Finland

{Antti.Pesonen, Antoni.Wolski}@vtt.fi
http://www.vtt.fi/tte/projects/rapid/

Abstract

In active monitoring, a monitoring system is given the responsibility of detecting conditions of interest, in a process, and taking the necessary actions like invoking alarms. RapidBase is a system for active monitoring utilizing an active database where database triggers, including fuzzy triggers, are used as basic monitoring tools. Two new instruments are introduced for use in fuzzy triggers: fuzzy quantifiers for expressing set-oriented propositions and fuzzy-temporal restrictors allowing to bind propositions to given intervals or points of time. The corresponding implementation is also described and its performance is analyzed.

Keywords: process management, control room, active database, fuzzy rule set, fuzzy trigger, fuzzy quantifier, fuzzy temporal restrictor.

1 Introduction

In industrial process management systems (PMS), the measurement data acquired from the process is scrutinized in a variety of ways. Aside from the role of the data in a closed-loop automatic control system (which is outside the scope of this work), the data is used to monitor the process in a control room environment. By a process we understand any continuous mechanized or computerized activity that may produce measurement data. Such processes and the corresponding management systems are common in traditional industries like chemical, petroleum, paper and energy industries. The concepts of SCADA (Supervisory Control and Data Acquisition) in the utilities industry and NMS (Network Management System) in telecommunications fall within the same category.

The scope of this paper is limited to the needs of short-time process monitoring. Typically, this involves control room personnel who interact with the system to receive the information needed to manage the system properly. Traditional functions of a PMS involve displaying current measurement values, measurement trends (i.e. time series

graphs) at different time densities and calculating various aggregate values (e.g. averages, minimums, maximums, etc.) over periods of time. More advanced requirements call for *active monitoring*. Active monitoring is a capability of a PMS to autonomously apply various computational and decision-making models to process data as it changes. This includes recognizing certain process states of various space and time complexity.

In a generalized implementation of a PMS with active monitoring capabilities, one would require to have:

(1) a high-level (meaning: easy) access to measurement data, both momentary and historical;

(2) active mechanisms to invoke computations based on data changes;

(3) decision-making models suitable for capturing process knowledge;

(4) model engines to run decision-making models;

(5) an easy-to-use way to define and modify data and decision-making models.

In our approach to the active monitoring problem, we have put together the ideas of temporal databases, active databases and fuzzy inference. The temporal database approach introduces a systematic treatment of all the data (point 1 above). The active database brings in data-oriented active mechanisms (2). The fuzzy inference model makes it possible to capture intuitive process knowledge handily (3), and the related fuzzy inference engine takes care of run-time model execution (4). The database approach, in general, makes it possible to use one high-level language to control and access all the objects involved (5). RapidBase is an active measurement database system based on the above principles. An overview of RapidBase is given in [WKLP99].

To our best knowledge, neither active nor temporal databases have been applied to the active monitoring problem. A related notion of ARCS (Active Rapidly Changing Data Systems) was introduced [Dat94], but no significant implementations has been reported. The temporal database research community (see [VK96], for a list of references) has concentrated on multitemporal data models and ignored mostly the need for a separate time series model. The support for time series has eventually received some attention [Dre94], and several implementations are known, but they are mostly tuned to the needs of financial applications.

Active databases were introduced [DM89] with the primary purpose of database integrity constraint maintenance. On the basis of input from our industrial customers, we extended the active database concepts to meet the monitoring requirements [WKP96] and, later on, we introduced *fuzzy triggers* [WB98]. In fuzzy triggers, the fuzzy inference model based on fuzzy rules [MJ94], and widely applied in fuzzy logic controllers (FLCs), was utilized.

The contribution of this paper is in the presentation of two new fuzzy instruments in the context of an observation database: *fuzzy quantifiers* for formulating propositions about observation value sets and *fuzzy temporal restrictors* for evaluation of fuzzy predicates in a temporal space.

The principles of using an active database for active monitoring are presented in Section 2. In Section 3, an introduction to fuzzy triggers is given. The model of fuzzy quantification in the context of active monitoring is included in Section 4. Fuzzy temporal restrictors are introduced in Section 5. Section 6 comprises implementation notes, and some performance results are presented in Section 7.

2 Applying active database to active monitoring

Active database is a database where data manipulation operations (such as an update) may ignite further calculations not explicitly specified by the issuer of the data manipulation operation. These further calculations are embodied in persistent database objects called active rules, or triggers. Users may specify triggers to achieve the required active behaviour of a database. In order to be able to do this, database languages like SQL have been enhanced to incorporate trigger definition syntax (see [WC96] for a survey of research in active databases).

The prevailing trigger execution model is called ECA (event-condition-action) [DM89]. An ECA trigger consist of three computational blocks that are executed sequentially and some of them conditionally:

- The *event block* is responsible for detecting an event of the type the trigger is tuned in to. Typically, an event type is a modification to a database table (an update or insert in SQL) or to a class extension or to an object instance. When a relevant event is detected, the trigger is said to be *fired*.
- The *condition block* is optional. If present, it specifies a predicate to be evaluated over the database, after the trigger is fired.
- The *action block* specifies the actions to be taken (launched) by the trigger when it is fired and the condition block evaluates to true. If the condition block is missing, the actions are launched upon trigger firing.

Triggers offer a power of expression needed in active monitoring. In the following example, a trigger called "boiling_alarm" is defined using the syntax of the RapidBase SQL-like language called RQL. The language keywords are shown in capital letters.

```
CREATE TRIGGER boiling_alarm UPDATE ON boilers
   WHEN OLD.temperature < 100 AND
        NEW.temperature >= 100
   DO SET state = 'boiling'
   DO CALL BoilingAlarm@AlarmDisplay (OTS, temperature);
```

The purpose of the trigger is to detect situations when the temperature of any boiler exceeds boiling point of 100^{o} C. The event type is an update on table "boilers". In the WHEN clause, the condition is specified to be evaluated for the triggering row, that is, a row in a table an update occurred in. By way of the OLD and NEW keywords, it is possible to access both the previous and the current column value in a triggering row, respectively. The DO clauses are used to specify the actions. The first one (SET) changes the value of a state variable in the triggering row. The second one (CALL)

sends a call to a remote procedure "BoilingAlarm" served by an external application named "AlarmDisplay".

In order to meet requirements of complex active monitoring, in RapidBase we have expanded the basic trigger model exemplified above. We introduced *timer triggers* [WKP96] to incorporate delays and, lately, *clock triggers* to utilize events generated by a real-time clock. Other contributions include [WKLP99]: multiple condition-action blocks allowing to specify a complex finite state automaton in a single trigger, trigger variables to facilitate execution time calculations, and various ways to utilize user-defined functions and procedures. Generally speaking, computations of any complexity, dealing with the wealth of the temporal measurement space, may be easily programmed by users and included both in the condition and action blocks.

3 Fuzzy triggers

This section introduces basic concepts of fuzzy sets and fuzzy inference [KY96], required to define fuzzy triggers . Also, the fuzzy trigger types of RapidBase are presented.

3.1 Basic concepts

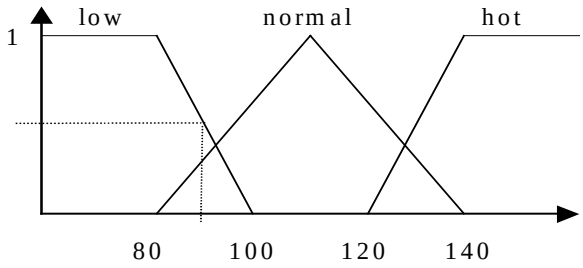
3.1.1 Fuzzy set

A *fuzzy set* is a set with imprecise boundaries in which the transition from membership to non-membership is gradual rather than abrupt. In this way, a fuzzy set F in a universe of discourse U is characterized by a *membership function* μ_F, which associates each element $u \in U$ with a grade of membership $\mu_F(u) \in [0, 1]$ in the fuzzy set F. Note that a classical set A in U is a special case of a fuzzy set with all membership values $\mu_A(u) \in \{0, 1\}$.

3.1.2 Linguistic variable

The basic concept underlying fuzzy logic is a *linguistic variable*, which is a variable whose values are words rather than numbers. A linguistic variable is characterized by a quintuple *(x, T(x), U, G, M)* in which *x* is the name of the linguistic variable; *T(x)* is the *term set* of *x*, that is, the set of names of linguistic values of *x* defined on *U*; *G* is a syntactic rule for generating the names of values of *x*; and *M* is a semantic rule for associating with each value its meaning.

Example: Let us consider the linguistic variable *Temperature*. Its term set *T(Temperature)* could be *T (Temperature) = {low, normal, hot}* where each term is characterized by a fuzzy set in a universe of discourse *U = [0, 300]*. We might interpret “low” as “a temperature below about 100°C,” “normal” as “a temperature close to 110°C,” and “hot” as “a temperature above about 120°C”. These terms can be characterized as fuzzy sets whose membership functions are shown in Fig. 1. For example, if the current temperature is 90°C then the *membership degree* to the fuzzy subset *low* is equal to 0.5.



μ_{low} = Trapezoidal(0, 0, 80, 100)
μ_{normal} = Trianglular(80, 110, 140)
μ_{hot} = Trapezoidal(120, 140, 300, 300)

Figure 1. The membership functions of the linguistic variable Temperature.

Fuzzy logic provides operations, which act on fuzzy sets. Those operations are counterparts to those, which act on crisp sets. For example, the union $(A \cup B)$ of two fuzzy sets is defined as:

$$\mu_{A \cup B}(x) = max\,(\mu_A(x)\,,\, \mu_B(x)) \;\forall\; x \in U$$

3.1.3 Fuzzy inference

The process of converting the crisp input data to a fuzzy set A*, is called *fuzzification*. It maps the input data into their membership functions. The most widely used fuzzifier is a *fuzzy singleton* defined by:

$$\mu_{A^*}(x) = 1 \qquad \text{if } x = x',\ \forall\ x \in U \tag{3.1}$$
$$\mu_{A^*}(x) = 0 \qquad \text{if } x \neq x'$$

In this case the fuzzy input set A* only contains the crisp element x'.

A *fuzzy implication* is viewed as describing a fuzzy relation between fuzzy sets forming the implication [MJ94]. A *fuzzy rule*, such as "if X is A then Y is B" is a fuzzy implication which has a membership function $\mu_{A \to B}(x, y) \in [0, 1]$. Note that $\mu_{A \to B}(x, y)$ measures the degree of truth of the implication relation between x and y. The *if* part of an implication is called the *antecedent (premise)*, whereas the *then* part is called the *consequent*. Using the *Mamdani's* (*minimum*) *implication*, the membership function of the fuzzy implication is defined as:

$$\mu_{A \to B}(x, y) = min[\mu_A(x), \mu_B(y)] \tag{3.2}$$

In fuzzy logic, Modus Ponens is extended to *Generalized Modus Ponens* in the following manner: given the input "X is A*" and the fuzzy rule "if X is A then Y is B" then the consequence is "Y is B*". The membership function of the conclusion, the fuzzy set B*, is defined as follows [Zad89]:

$$\mu_{B^*}(y) = \max_{x \in A^*}[\mu_{A^*}(x) \wedge \mu_{A \to B}(x, y)] \tag{3.3}$$

Generalized modus ponens has been adapted and used widely in control applications; the mechanism is called *interpolative reasoning*. This mechanism is needed for applications for which the input-output relationship is described by a collection of fuzzy if-then rules. Let us consider the following rule base (where X, Y and Z are linguistic variables defined on the universe of discourse U, V and W respectively).

$$R_i\text{: if } X \text{ is } A_i \text{ and } Y \text{ is } B_i \text{ then } Z \text{ is } C_i \quad i = 1..n$$

Given the crisp input (x_0, y_0), the goal is to determine the output "Z is C*" using fuzzy inference. The most commonly used fuzzy inference method in engineering applications is the so-called *Max-Min inference method* [Men95]. Inference steps used in RapidBase are explained in [WB98]

The result of the fuzzy inference system is a fuzzy set. The *defuzzyfication* step produces a representative crisp value as the final output of the system. There are several defuzzification methods [Men95]. The most commonly used is the *Centroid (Center-of-gravity) defuzzifier*, which provides a crisp value based on the center-of-gravity of the result (the output fuzzy set).

3.2 RapidBase fuzzy triggers

The concept of database triggers is enhanced in RapidBase to incorporate fuzzy inference. We proposed fuzzy triggers first in [BW96] (see also [BW97], [BKPW97] and [WB98]). There are two kind of fuzzy triggers: *C-fuzzy* and *CA-fuzzy triggers*. Both of these trigger types are based on the ECA trigger model. The example referenced to in the sequel can be found in the Appendix.

3.2.1 C-fuzzy trigger

In a C-fuzzy (Condition-fuzzy) trigger, a *fuzzy rule set* is encapsulated in a crisp-valued function, which can be used in an evaluation of a regular Boolean-valued predicate of the ECA trigger condition part. The C-fuzzy trigger provides a convenient way of introducing the fuzzy inference mechanism to traditional active database systems.

In RapidBase, we introduce a set of database objects for constructing a C-fuzzy trigger. A *linguistic type* encapsulates a set of terms to be used as a type of a linguistic variable (example command 1 in the Appendix). A fuzzy rule set is used by the *fuzzy inference engine* of the database Server to infer, ultimately, crisp results for the condition part of a trigger (example 3). To incorporate fuzzy reasoning into a C-fuzzy trigger, a fuzzy rule set call is used in the condition of the trigger (example 4).

3.2.2 CA-fuzzy trigger

A more powerful model, called a *CA-fuzzy trigger*, was also proposed [WB98] whereby approximate reasoning was integrated within the condition-action component of a trigger. In a C-fuzzy trigger we extended the condition part of a trigger with fuzzy predicates. In a CA-fuzzy trigger, a decision whether or not to execute a specific action is based on the fuzzy inference (hence fuzzy action).

The cause-and-effect relationship between the database state and concrete actions is expressed as a fuzzy rule set in the form:

```
if FP1 then FA1
if FP2 then FA2
...
if FPn then FAn
```

A fuzzy predicate FP_i is constructed from fuzzy propositions. FA_i is a fuzzy action proposition represented as: Z *is* z_i where Z is a linguistic variable having a set of linguistic terms $\{z_1, z_2, \ldots z_n\}$. We have also a set of concrete actions $A = \{a_1, a_2, \ldots a_n\}$ of which each a_i (i=1,2,...n) is a concrete action (e.g., a database command). The sets Z and A are said to be *mapped* to each other by associating z_i with a_i, for each i = 1, 2, ... n. Thus, each linguistic term z_i is uniquely associated with a concrete action a_i For example, in Fig. 2, the fuzzy actions: *zero*, *low*, *medium* and *high* are associated with the concrete actions $Action_1$, $Action_2$, $Action_3$ and $Action_4$, respectively.

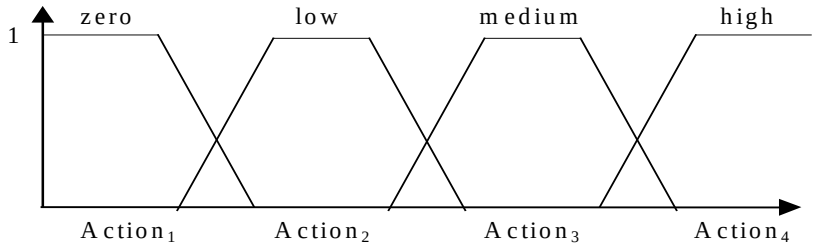


Figure 2. An example of membership functions of fuzzy actions.

The execution model of CA-fuzzy triggers involves detecting of an event, performing the interpolative reasoning process and executing an action in which the Center-of-gravity of the interpolative reasoning result has the highest membership degree. The term–action mapping can be seen in the example 5.

4 Quantified-fuzzy reasoning

Classical logical systems use two quantifiers: *universal* and *existential*. Fuzzy logic, on the other hand, admits a wide variety of *fuzzy quantifiers* [Pet96] exemplified by *few*, *several*, *usually*, *most*, *about ten*, etc. Fuzzy quantifiers are relevant in a database environment where we naturally maintain various structured sets of data (like tables, collections, etc.). A linguistically quantified proposition may be written as "Q A's ARE B" which means that Q elements of a set A are satisfying the fuzzy predicate B.

4.1 The calculus

To make the concept of a fuzzy quantifier meaningful, it is necessary to define a way of counting the number of elements in a fuzzy set or, equivalently, to determine its cardinality. The concept of *sigma-count* is used for this purpose: Let F be a fuzzy subset of $U = \{u_1, \ldots, u_n\}$ expressed symbolically as $F = \{(u, \mu_F(u)) / \forall u \in U\}$. The sigma-count of F is defined as the arithmetic sum of the membership degrees of the elements of F, i.e.:

$$\sum Count(F) \equiv \sum_i \mu_F(u_i) \quad i=1,\ldots,n \tag{4.1}$$

We need two steps to interpret the proposition Q A's ARE B. The first step is to compute the proportion of elements of A in B. The concept of *relative sigma-count*, denoted by $\sum Count(B/A)$, is used for this purpose and is defined as follows:

$$\sum Count(B/A) = \frac{\sum Count(B \cap A)}{\sum Count(A)} \tag{4.2}$$

The relative sigma-count of A in B can be easily expressed in terms of the membership functions of A and B. Particularly, when A is a crisp set, i.e. $\mu_A(u)=1 \; / \; \forall \; u \in A$, then the relative sigma-count becomes:

$$\rho = \sum Count(B/A) = \frac{\sum Count(B)}{Cardinality(A)} \tag{4.3}$$

The second step is to compute the truth of the proposition Q A's ARE B. It is defined as $\mu_Q(\rho)$; the membership degree of the result of the sigma-count, ρ, in the fuzzy set Q.

Example: Let us consider the proposition "*most motors ARE hot*". We'll assume the following information:

The value set of motor temperatures A={140, 130, 90, 150, 160}

The fuzzy quantifier *most* is defined by a TRIANGULAR(0.6, 0.8, 1)

The linguistic variable *hot* is defined by a TRAPEZOIDAL(120, 140, 300, 300)

The proportion of hot motors in a set of motors is computed using the above formula:

$$\rho = \frac{1+0.5+0+1+1}{5} = \frac{3.5}{5} = 0.7$$

Finally, the truth of the proposition "*most motors ARE hot*" is computed as follows:

$$\mu_Q(\rho) = \mu_Q(0.7) = \mathbf{0.5}$$

4.2 Quantified rule propositions in RapidBase

We include the construct *quantifier type* in RQL to support quantified sets. An example of a definition of a quantifier type can be found in the Appendix (example 6). Also, an example of a rule set with quantified fuzzy propositions, like "many motor_temperatures ARE hot" is included in the Appendix (example 7).

5 Temporal-fuzzy reasoning

A *fuzzy-temporal restrictor* is an entity that restricts a fuzzy proposition temporally. A temporally restricted fuzzy proposition can be written as 'A *be* B T' which means that A is satisfying the fuzzy predicate B, taking into account the temporal restrictor T. The temporal restrictor T is expressed as a membership function. *be* is a generic "be" verb which can take different tempus according to the rule. Let us look at few examples of temporally restricted fuzzy propositions:

```
motor WAS warm few_minutes_ago
chemical_balance HAS BEEN strongly_increasing lately
```

5.1 The calculus

Until now, we have been dealing with fuzzy qualifiers (e.g., 'hot') and quantifiers (e.g., 'most'). Temporal restrictor is a new concept and it needs to be formulated. At first, lets concentrate on the form of the proposition "A *be* B T" where A represents a single value (singular entity). Eventually, A can also be a set of values (e.g., in the rule *temperatures WERE hot recently*) possibly quantified with a fuzzy quantifier.

Lets look at an example of a membership function defining the temporal restrictor *few_minutes_ago* (Fig. 3a). The origin of the membership function graph describes the dynamic concept *now*, which means current time. The membership function *few_minutes_ago* is defined to have a positive membership degree between "six minutes ago" and "two minutes ago". Current time is changing all the time so, in fact, the membership function definition is also constantly changing; it is relative. For example, if the time was 04:00:15pm, *few_minutes_ago* was defined inside the time window 03:54:15pm and 03:58:15pm. Unlike with qualifiers and quantifiers, the evaluation time t_e of a temporally restricted fuzzy proposition has a significant effect on the result of the inference.

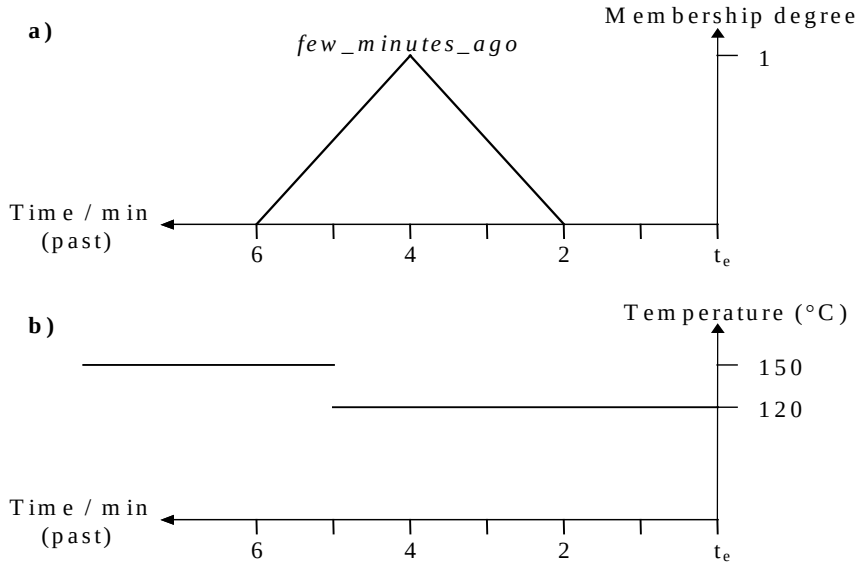


Figure 3. a) The membership function for few_minutes_ago, b) the motor temperature curve for the example motor.

Consider the example rule *motor WAS warm few_minutes_ago.* We need the history of temperatures of the motor in question, in order to be able to evaluate the rule. After fixing the evaluation time t_e, the definition area of the membership function of *few_minutes_ago* defines the time window, inside which we need to know the motor temperature. Let us consider an example where t_e is fixed (Fig. 3a and 3b). As you can see, the temperature value changes at t_e - 5 min when it drops from 150 °C to 120 °C.

Now, by combining the two diagrams above, we will get *temperature areas* for the evaluation of the membership function *few_minutes_ago*:

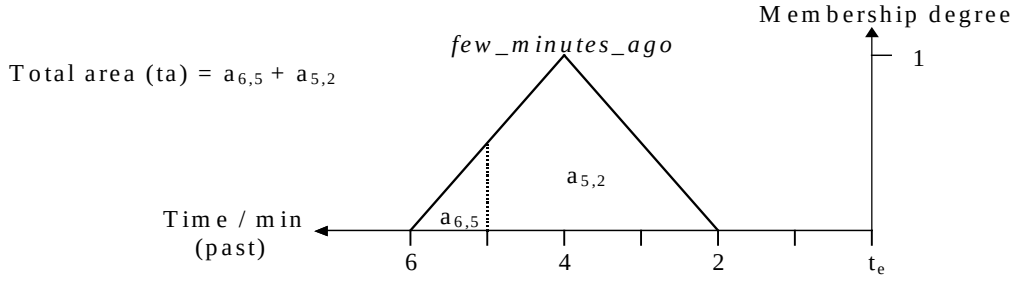


Figure 4. Temperature areas for the fixed example.

In the figure above, 'a' stands for *area*. So, $a_{6,5}$ is the area bounded by x-axis, the membership function segment and the line x = 5. Let us consider again the proposition *motor WAS warm few_minutes_ago.* At first, look at the time window (t_e - 6, t_e - 5). We can compute the partial truth of the proposition inside that window with $X_1 = \mu_{warm}(150)*(a_{6,5} / ta)$, where 'ta' stands for total area bounded by the membership function and x-axis. Similarly, the partial truth of the proposition inside the window (t_e - 5, t_e - 2) would be $X_2 = \mu_{warm}(120)*(a_{5,2} / ta)$. If the membership function for the term *warm* would give $\mu_{warm}(150) = 0.3$ and $\mu_{warm}(120) = 0.6$ then $X_1 = 0.3*(1/8) = 0.0375$ and $X_2 = 0.6*(7/8) = 0.525$.

We have now truth values for separate time windows inside the definition area of the membership function *few_minutes_ago*. To get the final truth value of the proposition *motor WAS warm few_minutes_ago* we simply take the arithmetic sum of the partial truth values. So `truth`(*motor WAS warm few_minutes_ago*) = 0.0375 + 0.525 = 0.5625.

Formally we can express the temporally restricted fuzzy proposition "A *be* B T"

$$\text{truth(A } be \text{ B T)} = \sum \left(\mu_B(meas(A_{i,i+1})) * \left(\frac{a^T_{i,i+1}}{ta^T}\right) \right), \forall i \in T \qquad (5.1)$$

where $meas(A_{i,j})$ means the value of the variable A inside the time window (i, j). $a^T_{i,j}$ declares the area bounded by the lines x = i, x= j and the membership function T. ta^T is a total area bounded by the function T and x-axis.

5.2 Unified framework

We have now three different concepts, which can be used to restrict a fuzzy result of a fuzzy inference using fuzzy rules. These are:

- fuzzy qualifiers, like *hot*, *strong* and *high*,
- fuzzy quantifiers, like *few*, *most* and *almost_all*,
- fuzzy temporal restrictors, like *lately* and *few_minutes_ago.*

We know how to evaluate a fuzzy proposition with fuzzy qualifier. We have introduced a method to evaluate fuzzy propositions with fuzzy quantifiers. We have also introduced a method to evaluate fuzzy propositions with fuzzy temporal restrictors. Now we need an evaluation method to calculate a fuzzy proposition with fuzzy quantifier and fuzzy temporal restrictor (e.g., the proposition *most motors WERE hot few_minutes_ago*). Formally, the combined fuzzy proposition can be expressed like this: "Q A's *be* B T" where Q is a fuzzy quantifier, A is a set of entities, B is a fuzzy qualifier and T is a fuzzy temporal restrictor. We can use a modified *relative sigma count* (ρ) (already presented in Section 3) to evaluate the first stage of the evaluation process of the above proposition.

$$\rho^{T} = \sum Count(B^{T} / A) = \frac{\sum Count(B^{T})}{Card(A)} \qquad (5.2)$$

The sigma count of $B^{T} = \sum Count(B^{T})$ is defined as the arithmetic sum of the membership degrees of the elements of B^{T}. With temporally restricted proposition those membership degrees are calculated using the formula 5.1.

The final result of the proposition "Q A's *be* B T" is given by the formula $\mu_{Q}(\rho^{T})$.

Example: Let us evaluate the fuzzy proposition *most motors WERE hot few_minutes_ago.* Motors and their temperatures are listed in the following tables:

motor1	ts	temp
	16:00:00	100
	16:00:01	110
	16:00:02	120
	16:00:03	130

motor2	ts	temp
	16:00:00	130
	16:00:01	140
	16:00:02	140
	16:00:03	140

motor3	ts	temp
	16:00:00	140
	16:00:01	150
	16:00:02	140
	16:00:03	130

The fuzzy quantifier *most*: TRIANGULAR (0.6, 0.8, 1.0).

The membership function *few_minutes_ago*: TRIANGULAR (2, 4, 6).

The linguistic variable *hot*: TRAPEZOIDAL (120, 140, 300, 300).

The evaluation time t_e of the proposition is fixed to be 16:00:06.

At first, we have to calculate the truth of *motor WAS hot few_minutes_ago* for each motor separately using the formula 5.1:

Motor1: $(\mu_{hot}(100)*(1/8)) + (\mu_{hot}(110)*(3/8)) + (\mu_{hot}(120)*(3/8)) + (\mu_{hot}(130)*(1/8))$ = 0*(1/8) + 0*(3/8) + 0*(3/8) + 0.5*(1/8) = 0 + 0 + 0 + 0.0625 = **0.0625**

Motor2: $(\mu_{hot}(130)*(1/8)) + (\mu_{hot}(140)*(3/8)) + (\mu_{hot}(140)*(3/8)) + (\mu_{hot}(140)*(1/8))$ = 0.5*(1/8) + 1*(3/8) + 1*(3/8) + 1*(1/8) = 0.0625 + 0.375 + 0.375 + 0.125 = **0.9375**

Motor3: $(\mu_{hot}(140)*(1/8)) + (\mu_{hot}(150)*(3/8)) + (\mu_{hot}(140)*(3/8)) + (\mu_{hot}(130)*(1/8))$ = 1*(1/8) + 1*(3/8) + 1*(3/8) + 0.5*(1/8)} = 0.125 + 0.375 + 0.375 + 0.0625 = **0.9375**

The proportion of motors, which were hot few minutes ago:

$$\rho^T = \frac{0.0625 + 0.9375 + 0.9375}{3} = \frac{1.9375}{3} \approx 0.65 \qquad (5.3)$$

Finally, the truth of the complete proposition *most motors WERE hot few_minutes_ago* is computed $\mu_Q(\rho^T) = \mu_{most}(0.65) = \mathbf{0.25}$

5.3 Temporal fuzzy reasoning in RapidBase

In RapidBase, we have implemented temporal fuzzy reasoning model presented above with one limitation: Currently, the measurement values are treated as one set. They are not grouped according to the real-world entities, like motors they correspond to. In the example above, this means that we have to process motor temperatures as a single set of values and not as three separate sets.

In RQL, we define a temporal restrictor type similarly to a linguistic type. For an example of such a definition, see the Appendix (example 8). To see a rule set with quantified and temporally restricted rule propositions, check the example 9 in the Appendix.

6 Implementation

The RapidBase Server is a C++ program that can be run on Windows NT, HP UNIX and Linux. The Server runs as a single operating system process. The overall architecture of the Server is presented in Fig. 5.

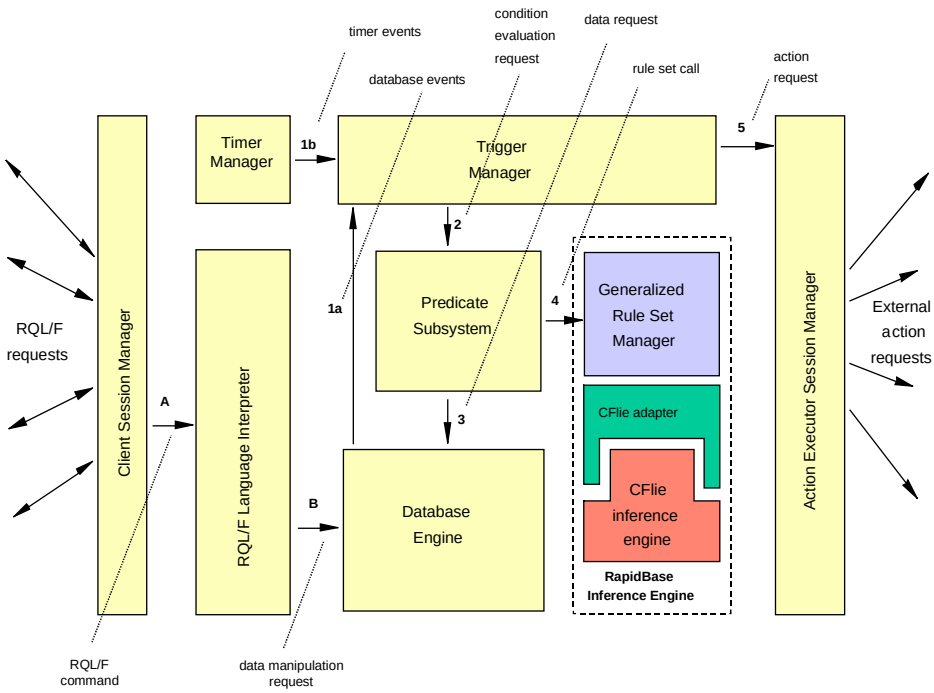


Figure 5. The RapidBase Server architecture.

The arrows shown in the figure illustrate the interaction between the sub-systems of the Server in trigger processing. A data manipulation command arrives from the network

and is passed to the interpreter for processing (A) and this, subsequently, results in elementary data requests executed by the Database Engine (B). A trigger is fired upon detection of a pre-defined event (a database event 1a or a timer event 1b). If there is a condition associated with the fired trigger, a stored predicate is invoked (2). The predicate uses the services of the Database Engine (3) to get the current data for the predicate evaluation. If a rule set call is specified, in the predicate, the corresponding rule set is invoked (4) with the necessary input data. The defuzzified rule set call result is returned to the predicate, which, in turn, returns the predicate evaluation result to the Trigger Manager. If the trigger condition is satisfied, the trigger actions are requested (5).

7 Performance

In this section we concentrate on the performance of the RapidBase Server and especially we address the efficiency of the inference engine of the Server. The test equipment used was an Intel Pentium III (533 MHz) PC running the Windows NT operating system. The whole database was fitted the main memory in order to avoid the virtual memory page swapping.

We forced updates as fast as possible to a history column of a table which resulted in about 3000 data updates per second. This means that one update took about 0.34 milliseconds to complete. The update time is taken into account when evaluating the inference engine efficiency. It is done by subtracting the update time from the average time used to process an update command activating the rule inference process. To avoid the influence of action execution on performance results, we also set the parameters of each test configuration to force no trigger action execution. We used only C-fuzzy triggers because both C- and CA-fuzzy triggers utilize the same inference engine, so the performance of the two in the sense of fuzzy inference is the same.

We ran tests with different type of rule propositions: 1) plain fuzzy proposition (e.g., motor IS hot), 2) quantified fuzzy proposition (e.g., most motors ARE hot) and 3) quantified-temporal fuzzy propositions (e.g., most motors HAVE BEEN hot lately). Each rule premise consisted of two propositions, connected with the AND operator. To show the effect on the performance caused by the rule set size, we used the rule sets of 2, 8 and 32 rules for each configuration. We varied also the quantified set size (10 and 100 entities) for quantified and quantified-temporal rules. Fig. 6 shows the results of the tests.

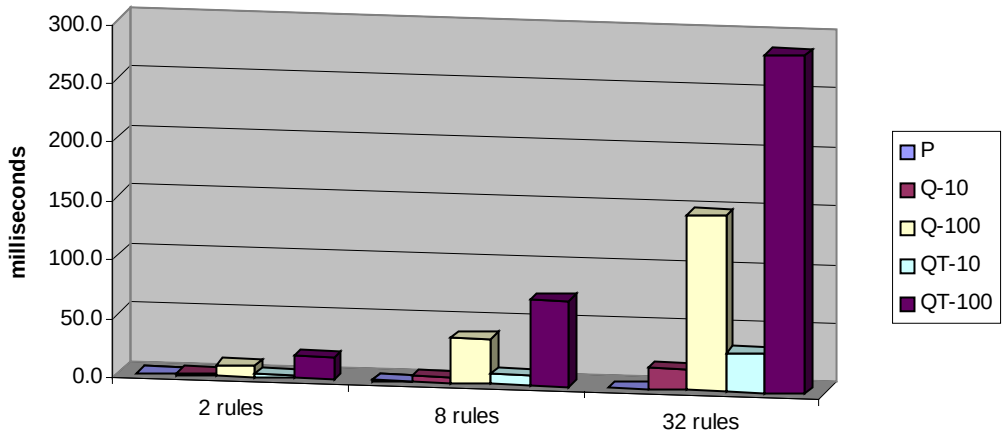


Figure 6. Execution time of fuzzy inference. Categories: P=Plain fuzzy proposition, Q= Quantified fuzzy proposition and QT=Quantified-temporal fuzzy proposition. The number in the category name states the size of the quantified set.

The increase in the size of a rule set does not have significant effect on the performance of the inference process of a plain (P) fuzzy rule (2 rules = 0.17 ms, 32 rules = 0.26 ms). For quantified (Q) and quantified-temporal (QT) rules the size of the rule set has a linear effect on the inference time: Increasing the rule set size by four times increases the evaluation time also about four times. Increasing the size of the quantified set (of values) has the same effect on the inference time.

8 Conclusions

We have demonstrated how an active database can be applied to active monitoring. Especially, by adding fuzzy inference capabilities, it is possible to increase the power of an expression of database triggers so that they are suitable to capture complex intuitive models in a concise form. Two new fuzzy instruments are introduced: fuzzy quantifiers and fuzzy temporal restrictors. Fuzzy quantifiers enable one to state set-oriented propositions that are a necessity in processing of huge measurement databases. Fuzzy temporal restrictors allow to restrict the propositions to various intervals and time points in the measurement history. The implementation of fuzzy mechanisms in RapidBase is also presented, together with the related performance figures.

References

[BKPW97] Bouaziz T. Karvonen J. Pesonen A. and Wolski A. Design and Implementation of TEMPO Fuzzy Triggers. *Proc. Eighth Int'l conference on Database and Expert Systems Applications (DEXA'97)*, Sept. 1-5, 1997, Toulouse, France, pp. 91–100, also at: http://www.vtt.fi/tte/projects/industrialdb/publs/publs.html.

[BW96] Bouaziz T. and Wolski A. Incorporating Fuzzy Inference into Database Triggers. Research Report No TTE1-2-96, VTT Information Technology,

Espoo, Finland, November 1996. Also at http://www.vtt.fi/tte/projects/industrialdb/publs/publs.html.

[BW97] Bouaziz T. and Wolski A. Applying Fuzzy Events to Approximate Reasoning in Active Databases. *Proc. Sixth IEEE Int'l Conference on Fuzzy Systems (FUZZ-IEEE'97)*, July 1-5, 1997, Barcelona, Catalonia, Spain, pp. 729–735. Also at: http://www.vtt.fi/tte/projects/industrialdb/publs/publs.html.

[Dat94] Datta, A. Research Issues in Databases for ARCS: Active Rapidly Changing Data Systems. *SIGMOD Record*, 23(3), September 1994, pp. 8–13.

[DM89] Dayal, U., McCarthy, D. The Architecture of an Active Database Management System. ACM SIGMOD Conference 1989, pages 215 - 224.

[Dre94] Dreyer, W. et al. Research Perspectives for Time Series Management Systems. *ACM SIGMOD Record*, 23(1), March 1994, pp. 10-15.

[KY96] George J. Klir and Bo Yuan (Eds.). Fuzzy sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers by Lotfi A. Zadeh. In *Advances In Fuzzy Systems-Applications and Theory*, Volume 6, 1996, 821 pages.

[Men95] Mendel J. M. Fuzzy Logic Systems for Engineering: A Tutorial. In Proc. of the IEEE, Special Issues on Engineering Applications of Fuzzy Logic, Vol. 83, No. 3, March 1995, pp. 345 - 377.

[MJ94] Munakata T., Jani Y. Fuzzy Systems: An Overview. *Communications of The ACM*. Vol. 37, No. 3, March 1994, pp. 69 - 76.

[Pet96] Petry F. E. Fuzzy Databases; Principles and Applications. Kluwer Academic Publishers, Boston, 1996.

[VK96] Vassilis J., Kumar, Tsotras and Anil . Temporal database bibliography update. *ACM SIGMOD Record*, 25(1), March 1996. pp. 41-51.

[WB98] Wolski, A. and Bouaziz, T. Fuzzy Triggers: Incorporating Imprecise Reasoning into Active Databases. *Proc. 14th International Conference on Data Engineering (ICDE'98)*, Feb. 23-27, 1998, Orlando, Florida, pp. 108-155. Also at http://www.vtt.fi/tte/projects/industrialdb/publs/publs.html.

[WC96] Widom, J. and Ceri, S. (eds.). Active Database Systems: Triggers and Rules For Advanced Database Processing. Morgan Kaufmann, 1996.

[WKLP99] Wolski, A., Kuha, J., Luukkanen, T. and Pesonen, A. Design of RapidBase—an Active Measurement Database System. Research Report TTE1-5-99, VTT Information Technology, Espoo, Finland, October. Also at http://www.vtt.fi/tte/projects/industrialdb/publs/publs.html.

[WKP96] A. Wolski, J. Karvonen and A. Puolakka, "The RAPID Case Study: Requirements for and the Design of a Fast-Response Database System", *Proc. First Workshop on Real-Time Databases (RTDB'96)*, March 7-8, Newport Beach, CA, USA, pp. 32–39, also at http://www.vtt.fi/tte/projects/industrialdb/publs/publs.html.

[Zad89] Lofti A. Zadeh Knowledge Representation in Fuzzy Logic. In *IEEE Transactions on Knowledge and Data Engineering*, 1(1), 1989, pp. 89-100.

Appendix

This appendix contains the RQL commands used as examples in the paper.

```
(1)  CREATE LINGUISTIC TYPE Temperature INTEGER (
       low         TRAPEZOIDAL (0, 0, 80, 100),
       normal      TRIANGULAR (80, 110, 140),
       hot         TRAPEZOIDAL (120, 140, 300, 300)
     )

(2)  CREATE LINGUISTIC TYPE Severity INTEGER (
       alarm_none        TRAPEZOIDAL (0, 0, 0.5, 1),
       alarm_low         TRAPEZOIDAL (0.5, 1, 1.5, 2),
       alarm_medium      TRAPEZOIDAL (1.5, 2, 2.5, 3),
       alarm_high        TRAPEZOIDAL (2.5, 3, 4, 4)
     )

(3)  CREATE RULE SET ControlAlarm
     (temperature Temperature, torque Torque)
     Severity DEFAULT alarm_none (
       IF temperature IS hot AND torque IS medium THEN alarm_low,
       IF temperature IS hot AND torque IS high THEN alarm_medium
     )

(4)  CREATE TRIGGER MotorTemperature
     UPDATE OF hist.temp ON motors
     WHEN ControlAlarm(hist.temperature, hist.torque) > 2
     DO CALL HighTemp@Actions USING hist.temperature, hist.torque

(5)  CREATE OR REPLACE TRIGGER MotorTemperature
     UPDATE ON motors
     DO INFER FROM ControlAlarm(hist.temperature, hist.torque)
     ACTIONS
     (
            WHEN alarm_low          DO CALL alarmLow@actionServer,
            WHEN alarm_medium       DO CALL alarmMed@actionServer,
            WHEN alarm_high         DO CALL alarmHigh@actionServer
     )

(6)  CREATE RELATIVE QUANTIFIER TYPE Amounts FLOAT (
     few     TRAPEZOIDAL (0, 10, 30, 40),
     many    TRAPEZOIDAL (30, 50, 70, 80),
     most    TRAPEZOIDAL (70, 80, 100, 100)
     )

(7)  CREATE OR REPLACE RULE SET ControlAlarm
     (motor_temps Temperature COLLECTION QUANTIFIED WITH Amounts)
     Severity DEFAULT alarm_none (
     IF many motor_temps ARE hot THEN alarm_low,
     IF most motor_temps ARE hot THEN alarm_high
     )

(8)  CREATE TEMPORAL RESTRICTOR TYPE minute_based_restr MINUTE (
     few_minutes_age          TRIANGULAR (2, 4, 6),
     several_minutes_agoTRIANGULAR (4, 10, 16),
     long_time_ago       TRIANGULAR (10, 16, 100, 100)
     )

(9)  CREATE OR REPLACE RULE SET ControlAlarm
     (motor_temps Temperature HISTORY QUANTIFIED WITH Amounts
     TEMPORALLY RESTRICTED WITH minute_based_restr)
     Severity DEFAULT alarm_none (
     IF most motor_temps HAVE BEEN hot few_minutes_ago THEN alarm_low,
     IF most motor_temps HAVE BEEN hot several_minutes_ago THEN alarm_medium
     )
```