VTT INFORMATION TECHNOLOGY RESEARCH REPORT TTE1-2-96

TEMPO Project

Incorporating Fuzzy Inference into Database Triggers

Tarik Bouaziz
Antoni Wolski

Abstract

Triggers (active rules) bring domain-specific reactive behavior to databases. Traditional Event-Condition-Action triggers use only crisp concepts. In this paper, fuzzy triggers are proposed whereby approximate reasoning may be integrated with a traditional crisp database. The new approach paves the way for intuitive expression of application semantics of imprecise nature in database-bound applications. Several levels of applying fuzzy concepts to triggers are proposed. Fuzzy propositions may be incorporated, as elementary predicates, in Boolean trigger conditions. A significant improvement of the condition expression power is achieved by encapsulating a set of fuzzy rules into a Boolean-valued function called the rule set function. Subsequently, actions are expressed also in fuzzy terms, in a trigger model. Examples are provided to illustrate how fuzzy triggers can be applied to a real-life drive control system in an industrial installation.

Espoo, November 1996

VTT INFORMATION TECHNOLOGY
Printed 11/20/96

P.O. Box 1201
FIN-02044 VTT, FINLAND

Tel. (+358 9) 4561
Fax (+358 9) 456 6027

I. Introduction

Active databases which incorporate Event-Condition-Action rules into the conventional (passive) databases have been investigated by many researchers over the past decade [VK91, PDW+93, DHW94, FT95, CW96b]. They provide the capability to react to database (and possibly external) stimuli, called *events*, without user intervention. The most predominant direction of the previous research has been the development of knowledge and execution models and the investigation of architectural issues (see [CW95a, p. 303-324] for a reference list). Another research direction is characterized by intersection of other research fields, for instance temporal or real-time databases, and active databases [BH95, RSS+96]. We are investigating another possibility: how to incorporate the fuzzy logic paradigm into active databases.

Fuzzy logic [Zad65] deals with statements that can be true to a certain degree: the values are between 1 (completely true) and 0 (completely false). Thus, an objective of fuzzy logic (FL) is to provide a systematic basis for representing imprecision and/or uncertainty. Another objective of FL is to mimic the ability of the human mind to effectively employ modes of reasoning that are *approximate* rather than exact. Nowadays, applications of fuzzy logic are found in many fields [MJ94] including artificial intelligence (e.g., expert systems), databases (e.g., information retrieval), robotics, pattern recognition, decision analysis, diagnostics, etc.

Applying fuzzy logic to databases has been an active research area since the 80's [DEB89, VSC+89, Pet96]. The most important issues which have been addressed are i) the enhancement of existing data models for representing uncertain and/or imprecise data (fuzzy data), ii) the extension of current database languages (e.g., SQL) to handle fuzzy queries, and iii) the use of fuzzy inference to deduce answers to questions in fuzzy expert database systems [ZK84, Zem89, LWL89].

To our best knowledge, no attention has been paid until now on integrating imprecision and/or uncertainty within database triggers. Similar work has been addressed in the context of fuzzy expert systems [Zad84, Zad89]. Zadeh noted that since most expert's knowledge is fuzzy, most of its facts and rules are fuzzy. Typically, a knowledge base is built through consultations with human experts, whose expertise and knowledge are invariably imprecise, incomplete, or not totally reliable. Existing expert systems, he noted, ignore the fuzziness of such information. Fuzzy expert systems allow fuzziness of antecedents and/or consequents in the rules of the form "if X is A then Y is B" where "X is A" and "Y is B" are fuzzy propositions. Fuzzy modifiers and quantifiers can be used in the antecedent and/or consequent of a rule. In addition, fuzzy expert systems are based on approximate reasoning [GKB+84]. Approximate reasoning deals with inference under imprecision and/or uncertainty in which the underlying logic is approximate rather than exact.

The main idea behind our work is similar in spirit to the one considered in the context of fuzzy expert systems[1]. It is widely accepted that the volume of data stored in databases is ever-increasing. This phenomenon, referred to as "information explosion", characterizes many applications. Tools are needed to transform these data to *useful* information that enables users to make decisions. The work presented in this paper was originally driven by the requirements of a real application domain which was a paper machine drive control system. Users wanted a single mechanism which integrates two of their needs. First, they wanted to have the ability to use data in a fuzzy manner which is close to their thinking. On the other hand, they wanted to be warned when the database reach situations of interests, for example to avoid failures of the drive system. Thus, merging fuzzy logic features and active database capability is just a natural step to go on.

The incorporating of imprecision and/or uncertainty within database triggers may concern the event, the condition or the action components of an active rule or all together. We will not elaborate further on the

[1] Note that several active database features have been adopted from production rule systems [BD83].

specific semantics of different possible dimensions expressing the behavior of active database systems [PDW+93, FT95]. The contribution of this paper is twofold:

- The presentation of a simple approach of integrating fuzzy rules and fuzzy inference with database triggers. The purpose is to highlight how a conventional active database system can be easily extended with fuzzy concepts.
- The development of more general fuzzy trigger models which focus on the definition of fuzzy actions and fuzzy events. The purpose of these models is to identify further levels of integrating fuzzy concepts with active rules.

The condition component of a conventional active rule specifies what must be checked once the rule is triggered and before the action is executed [WC96a]. A condition is a predicate, like the SQL where-clause, on the database state. However, existing active rule languages do not consider fuzzy predicates which provide a higher level of abstraction than crisp conditions. Fuzzy predicates are based on terms like *young*, *rich*, *high*, *tall*, etc. which allow the representation of vagueness. In addition, fuzzy predicates can be used in fuzzy rules leading to application of fuzzy inference [Zad89]. We propose to embed fuzzy predicates and fuzzy rules within the Boolean-valued trigger condition. Model of such a trigger, called a C-fuzzy trigger, is presented in Section 3.

A fuzzy action, in the context of fuzzy databases (which store fuzzy data), is based on data manipulation operations on fuzzy attributes. However, fuzzy databases are not widely used, so we assume conventional (non fuzzy) databases in this work. A fuzzy action, defined by a linguistic term, can be fired partially and the decision to execute an action is based on an approximate reasoning process which aggregates the overlapping of elements of fuzzy actions. The corresponding CA-fuzzy triggers is proposed in Section 4.

Notes on implementations are presented in Section 5. We conclude by underlying some open research problems.

I. Basic concepts of fuzzy sets and possibility theory

This section introduces basic concepts of fuzzy sets and possibility theory [Zad65, Zad78, DP88]. Informally, 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 $\mu_A(u) \in \{0, 1\}$.

A. Linguistic variables

Linguistic variables are variables whose values are not numbers but words or sentences in a natural language. A linguistic variable is usually decomposed into a set of *terms* (linguistic values). Let us consider a fuzzy set A of people whose age belongs to the interval [0, 120], a linguistic variable *Age* which is decomposed into a set of linguistic values, denoted by T, such as T(Age) = {Young, Middle-aged, Old}. The meaning of such values is defined by their membership function (Fig. 1.) :

μ_{Young} = Trapezoidal(0, 0, 20, 30)
$\mu_{Middle\text{-}aged}$ = Trapezoidal(25, 30, 40, 45)

μ_{Old} = Trapezoidal(40, 60, 120, 120)

Figure 1. Membership functions of the linguistic variable Age.

For example, if John is 22 years old then the membership degree to the fuzzy subset *Young* is equal to 0.7. The fuzzy proposition is then represented by "Age(x) is Young", x ∈ A, which truth is matter of degree.

A. Fuzzy operations

Fuzzy logic provides operations, that act on fuzzy sets, that are counterparts of those which act on crisp sets. Fuzzy set operators for union, intersection, equality, and complementation are defined as follows:

Union: The union $(A \cup B)$ of two fuzzy sets is defined as $\mu_{A \cup B}(x) = max\,(\mu_A(x)\,,\, \mu_B(x)) \;\forall\; x \in U$	*Intersection:* The intersection $(A \hat{O} B)$ of two fuzzy sets is defined as $\mu_{A \hat{O} B}(x) = min(\mu_A(x)\,,\, \mu_B(x)) \;\forall\; x \in U$
Equality: Two fuzzy sets A and B are equal iff $\mu_A(x) = \mu_B(x) \;\forall\; x \in U$	*Complement:* A´ is complement of A iff $\mu_{A'}(x) = 1 - \mu_A(x) \;\forall\; x \in U$

A. Quantifiers and modifiers

Classical logical systems use two quantifiers: *universal* and *existential*. Fuzzy logic, as for it, admits a wide variety of fuzzy quantifiers exemplified by *few*, *several*, *usually*, *most*, *about ten*, etc. Fuzzy propositions with quantifiers belong to one of the two following classes:

1. "Q X's are A" means Q elements of a set X are satisfying the fuzzy predicate A. For example, "most basketball-players are tall" where *tall* is a linguistic value defined by a membership function.
2. "Q X's B are A" means Q elements of a set X which satisfy the fuzzy predicate B also satisfy the fuzzy predicate A. For example, "most young employees have a small salary" where *young* and *small* are linguistic values defined by membership functions.

Linguistic hedges or *modifiers* provide a means to modify membership functions to apply more, or less, stringent criteria to the fuzzy set membership functions. In other words, they affect the membership functions by intensifying (concentrating) or spreading (dilating) its shape. Terms such as *very*, *extremely*, *somewhat*, *more-or-less*, *quite* are examples of modifiers.

A. Possibility distribution

Possibility theory is based on fuzzy set theory and it provides a unified framework to deal with information both imprecise and/or uncertain [Zad78, Zad89, DP88, BP93]. A concept which plays a central role in possibility theory is that of a *possibility distribution*. If X is a linguistic variable taking values in a universe of discourse U, then the possibility distribution of X, Π_X, is the fuzzy set of all possible values of X. More specifically, let "X is F" be a fuzzy proposition where F is a fuzzy set defined by a membership function μ_F, then the possibility distribution Π_X, is equal to F. The equation "Π_X= F" means that the degree of possibility which the given variable X takes *u* as a value is equal to the membership value of *u* in the fuzzy set *F*, i.e.

$$\Pi_X(u) = \mu_F(u) \;\forall\; u \in U$$

For example, if we know only that "John is young" (but not his precise age), where the meaning of *Young* is described by a membership function μ_{young}, the possibility that the age of John takes u is defined as follows: $\Pi_{Age(John)}(u) = \mu_{young}(u) \ \forall u \in$ Dom(Age).

A. Fuzzy inference

A *fuzzy implication* is viewed as describing a fuzzy relation between the fuzzy sets forming the implication [MJ94]. A fuzzy rule, such as "if X is A then Y is B" is implemented by a fuzzy implication (fuzzy relation) which has a membership function $\mu_{A \rightarrow B}(x, y) \in [0, 1]$. Note that $\mu_{A \rightarrow B}(x, y)$ measures the degree of truth of the implication relation between x and y. In control applications, *Mamdani implication* (minimum) is one of the most commonly used interpretation for the implication and it is defined as:

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

Note that this implication is chosen because it traduces the cause and effect relationship [Men95]. In the case of active databases, the Condition-Action components of an active rule traduces also a cause and effect relationship.

In fuzzy logic, Modus Ponens is extended to Generalized Modus Ponens in the following manner: if "X is A*" and "if X is A then Y is B" then "Y is B*". The membership function of the conclusion, the fuzzy set B*, is defined as follows [Zad84]:

$$\mu_{B*}(y) = max_{x \in A*}[\mu_{A*}(x) \wedge \mu_{A \rightarrow B}(x, y)]$$

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 relationships is described by a collection of fuzzy if-then rules*. In addition, Interpolation serves to minimize the number of if-then rules [Zad89]. Let us consider a rule base (where X, Y and Z are linguistic variables):

R_i: *if X is* A_i *and Y is* B_i *then Z is* C_i $i = 1..n$

and given the input fact (x_0, y_0), the goal is to determine the output "Z is C*". A fuzzy logic system uses the following steps to achieve this goal:

1. Fuzzification:

The fuzzifier maps the input data $x' = x_0 \in U$ into a fuzzy set A*. The most widely used fuzzifier is a fuzzy singleton defined by:

$\mu_{A*}(x) = 1$ if $x = x', \forall x \in U$

$\mu_{A*}(x) = 0$ if $x \neq x'$

When the fuzzy input set A* only contains a crisp element x', (1) becomes:

$$\mu_{B*}(y) = 1 \wedge \mu_{A \rightarrow B}(x', y) = \mu_{A \rightarrow B}(x', y) \qquad (2)$$

1. Interpolative reasoning (e.g., the Max-Min Inference method):

The truth value for the premise of each rule is computed, and applied to the conclusion part of each rule. The membership functions defined on the input variables are applied to their actual values to determine the degree of truth for each rule premise. The degree of truth for a rule's premise is sometimes referred as its *alpha*. It is computed as follows:

$$\alpha_i = \mu_{Ai \text{ and } Bi}(x_0, y_0) = \min(\mu_{Ai}(x_0), \mu_{Bi}(y_0))$$

If a rule's premise has nonzero degree of truth then the rule is activated. The second step is to find the output, C'_i, of each of the rules:

$\mu_{C'i}(w) = \mu_{(Ai \text{ and } Bi) \rightarrow Ci}(x_0, y_0, w), \forall w \in W$

In Min inferencing (or Mamdani implication rule) the implication is interpreted as a fuzzy *and* operator:

$\mu_{C'i}(w) = \mu_{(Ai \text{ and } Bi) \rightarrow Ci}(x_0, y_0, w), \forall w \in W$

$= \mu_{Ai \text{ and } Bi}(x_0, y_0)$ and $\mu_{Ci}(w) = \min(\mu_{Ai \text{ and } Bi}(x_0, y_0), \mu_{Ci}(w))$

In the composition subprocess, all fuzzy subsets assigned to each output variable are combined together to form a single fuzzy subset for each output variable. The purpose is to aggregate all the individual rule outputs to obtain the overall system output. In the Max composition, the combined output fuzzy subset C* is constructed by taking the maximum over all of the fuzzy subsets assigned to the output variable by the inference rule:

$\mu_{C*}(w) = \max(\mu_{C'1}(w), \mu_{C'2}(w), \ldots, \mu_{C'n}(w))$

1. Defuzzification:

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], for example the *Centroid (Center-of-gravity) defuzzifier* provides a crisp value based on the center-of-gravity of the result (the output fuzzy set).

I. C-fuzzy triggers

In this section we propose a trigger model incorporating fuzzy predicates and the interpolative reasoning technique introduced above, in the process of the evaluation of the condition part of an ECA trigger. We are calling such triggers *C-fuzzy triggers* (or Condition-fuzzy ECA triggers). Fuzzy predicates add a higher-level abstraction to triggers. Besides incorporating approximate reasoning within triggers, the interpolative mechanism could be useful to reduce the proliferation of triggers which can affect the performance of the system. In addition, the execution model of C-fuzzy triggers can be easily implemented, as existing fuzzy inference tools can be used.

A. The basic ECA trigger condition model

We assume simple ECA trigger execution model ([WC96b], p. 17). In algorithmic terms, it consists of three consecutive calculations performed when an event occurs:

a) detecting of an event and finding of a relevant trigger,

b) evaluating the condition and

c) executing the action if the condition is true.

The above algorithmic components are briefly called the event, condition and action parts, respectively.

Let e_i be an event occurring at time t_i. The event time is atomic and no two events can occur at the same time point. The time of the preceding event, e_{i-1}, can be thus denoted by t_{i-1}. The database state, after an event e_i has occurred, is the current sate S_i and the state before the event occurred—the previous state S_{i-1}.

The condition C evaluated at time t_i is a Boolean predicate defined over the current and the previous states:

$$C_{t_i} : \{S_i \times S_{i-1}\} \rightarrow \{0, 1\}$$

The condition may be represented as an expression composed of predicates, elementary predicates and Boolean operators AND, OR and NOT, and parentheses for precedence.

A typical elementary predicate is a comparison predicate defined over all the database values in the predicate domain, the comparison connectors and legal literals:

$CP_{t_i} : \{VAL \times COMP \times LIT\} \rightarrow \{0,1\}$, such that:

$CP(val, comp, lit) = 1$ if *val comp lit* is true
$CP(val, comp, lit) = 0$ if *val comp lit* is false

where $val \in VAL$ is a value (e.g. of a data item) defined over S_i or S_{i-1}, $comp \in COMP$ is a comparison connector like = , >, <, etc., and $lit \in LIT$ is a literal. In typical trigger implementations, *val* is limited to an expression defined over items of the database object the event is related to, like a row in a table, in a relational database. Aggregate functions are usually disallowed for performance reasons [WC96a].

A. Incorporating fuzzy predicates

We extend the conventional condition model by introducing a new type of a (Boolean) elementary predicate expressed as a fuzzy predicate. A fuzzy predicate, such as (X is A) applies the fuzzy term *A* to the linguistic variable *X*. The evaluation of the fuzzy predicate is based on the computation of a *degree of match*. A degree of match, between a current value *v* and a fuzzy proposition (X is A), is defined as a membership degree of *v* belonging to the fuzzy set *A*. Then, if the degree of match is greater than zero (i.e. $\mu_A(v)>0$) then the predicate evaluation returns *true*, otherwise *false*. For example, the following statement on a motor *"the temperature of a motor m1 is high"* can be expressed by the following fuzzy predicate:

Temperature(m1) is High

Let us assume the current temperature of *m1* is *t*. The evaluation of this predicate is as follows: a degree of match, between the current temperature and the fuzzy proposition, is computed. If $\mu_{High}(t)>0$, the predicate returns *true*.

A more complex fuzzy proposition could be defined using the *where-clause* of a fuzzy database query language (e.g., SQLf [BP95]).

A. Interpolative condition model

In order to utilize the expressive power of fuzzy rules and to apply fuzzy inference to the trigger condition part, we propose a special function called the *rule set function* (RSF) in the comparison predicate:

$RSF_t : \{R \times S_i \times S_{i-1}\} \rightarrow D$

where *R* is a set of fuzzy rules (fuzzy implications), each of which is in the form:

if FP then FC

where *FP* is a fuzzy antecedent (predicate) and *FC* is a fuzzy consequent [Zad89]. *FP* is constructed from fuzzy propositions

q_x *X IS|ARE* a_x

connected by fuzzy operators *and*, *or* and *not*. *X* is a linguistic variable representing a database value fuzzyfied using the terms (and the corresponding membership functions) in the term set A_x, $a_x \in A_x$. The optional fuzzy quantifier $q_x \in Q_x$ (Q_x is a set of quantifier terms of *X*) may be also used.

FC is in the form:

Y IS b_y

where *Y* is the consequent linguistic variable (the same one occurring in all the fuzzy rules in *R*) and $b_y \in B_y$ is a term of the consequent variable.

The range of RSF, *D,* is a domain (universe of discourse) of the linguistic variable Y. Thus, RSF yields a value which can be used in a regular comparison predicate evaluating, in turn, to *true* or *false*.

An example of a fuzzy rule (without quantifiers) may be the following:

```
if temperature IS high AND pressure IS very_high
then alarm IS very_high
```

where `temperature` and `pressure` are linguistic variables representing fuzzified values of the crisp data items, for example "temp" and "press", and `alarm` represents an output value of the rule set function.

Fuzzy quantifiers, although not much used in control applications, are very relevant in a database environment where we naturally maintain various structured sets of data (like tables, classes, collections, extensions, etc.). An example of applying a quantifier *most* to a rule could be:

```
if most motors ARE hot AND temperature IS rising
then set_alarm IS high
```

The evaluation of the rule set function is as follows:

1) the linguistic variables in the antecedent part of the rules are evaluated, i.e. the corresponding source data are fuzzyfied (we are assuming no fuzzy values are stored in the database),
2) the interpolative reasoning method, for example the Max-Min method, is applied to rules, producing a fuzzy conclusion,
3) the conclusion is defuzzified using, for example, the Center-of-gravity method to yield the crisp function value which is then applied to the comparison predicate.

In addition to being able to define rule set functions, one must be able to define linguistic variables and the corresponding terms, in the system.

A. Example: how to generate an overheating alarm in a drive system

The following example is based on the case study provided by ABB Industry Oy, a manufacturer of complex drive systems for industrial installations. The example pictures a paper machine equipped in tens of high-power electric motors running at different speeds and loads. The problem we are illustrating here is how to generate a synthetic alarm information about motor overheating in a system like that. A serious overheating of a single motor may be a cause for an alarm to the same extend as a moderate overheating of a number of motors. It is required to have alarms reported at different intensity levels (say from 1 to 4) depending on the seriousness of the situation. The dynamics of the system has also to be taken into account, i.e. the fact of the rising temperature implies a higher alarm level than that of the decreasing temperature. We shall show how this complex task can be achieved by way of just *one* C-fuzzy trigger. The syntax of the prototype language RQL/F (RapidBase Query Language/Fuzzy) is used in this example (see implementation notes in Section 6).

Assume that all the relevant motor measurement data are stored in a single (possibly temporal) table having the following schema:

```
motor( motorId, temp, deltaTemp )
```

where temp is the measured values of the motor's temperature and the derived column deltaTemp represents the difference between the current and the previous temperature reading, normalized by diving by the previous reading.

In order for membership function definitions to be reusable in a database, the concept of a linguistic type is introduced. A linguistic type embodies a set of compatible terms to be applied to various linguistic variables.

The following linguistic types are used to represent fuzzified values of temperature:

```
CREATE LINGUISTIC TYPE Temperature INTEGER(
     normal    TRAPEZOIDAL (0, 0, 120, 140),
     hot       TRAPEZOIDAL (120, 140, 300, 300),
     very_hot  TRAPEZOIDAL (145,160, 300, 300)
)
```

The next one is a general linguistic type to deal with values whose interesting value range is [0, 1]:

```
CREATE LINGUISTIC TYPE NegativeToPositive FLOAT(
     negative TRAPEZOIDAL (-1, -1, -0.4, -0.2),
     big_negative TRAPEZOIDAL (-1, -1, -0.8, -0.6),
     small_negative TRAPEZOIDAL (-0.8, -0.6, -0.4, -0.2),
     zero TRAPEZOIDAL (-0.4, -0.2, 0.2, 0.4),
     small_positive TRAPEZOIDAL (0.2, 0.4, 0.6, 0.8),
     big_positive TRAPEZOIDAL (0.6, 0.8, 1, 1),
     positive (0.2, 0.4, 1, 1)
)
```

The last linguistic type will be used to represent the alarm severity level:

```
CREATE LINGUISTIC TYPE AlarmSeverity FLOAT(
     none      TRAPEZOIDAL (0, 0, 0.5, 1.0),
     low       TRAPEZOIDAL (0.5, 1.0, 1.5, 2.0),
     medium    TRAPEZOIDAL (1.5, 2.0, 2.5, 3.0),
     high           TRAPEZOIDAL (2.5, 3.0, 4.0, 4.0)
)
```

Fuzzy quantifiers are also classified as types:

```
CREATE QUANTIFIER TYPE Amounts (
     few  TRAPEZOIDAL (0, 0, 20, 30),
     some      TRAPEZOIDAL (20, 30, 60, 70),
     most      TRAPEZOIDAL (60, 70, 100, 100)
)
```

Quantifiers may be applied to value sets yielding fuzzy quantified sets. The first one is a set of motor temperatures:

```
CREATE VALUE SET motorTemperatures OF
     ( SELECT temp FROM motor)
```

and the second one is a set of temperature deltas:

```
CREATE VALUE SET motorTempDeltas OF
     ( SELECT deltaTemp FROM motor )
```

Now, a rule set is defined. The rule set definition has the form of a function taking parameters and returning a crisp value. In the following definition, the rule set OverheatingAlarmLevel takes two quantified set parameters of types motorTemperatures and motorTempDeltas, respectively, it uses the membership functions of linguistic

type AlarmSeverity in the rule consequent, and it returns the default (defuzzified) value of *none* if no rule is fired:

```
CREATE RULE SET OverheatingAlarmLevel (
     motors Temperature QUANTIFIED WITH Amounts,
     deltas NegativeToPositive QUANTIFIED WITH Amounts
)
AlarmSeverity DEFAULT none (
     IF some motors ARE hot
          AND most deltas ARE big_positive
          THEN low,
     IF some motors ARE very_hot
          AND some deltas ARE big_positive
          THEN low,
     IF some motors ARE very_hot
          AND most deltas ARE big_positive
          THEN medium,
     IF most motors ARE hot
          AND some deltas ARE big_positive
          THEN medium,
     IF most motors ARE hot
          AND most deltas ARE big_positive
          THEN high,
     IF most motors ARE very_hot
          THEN high
)
```

The above rules set evaluates to a float-valued alarm level based on the amount of overheated motors, the degree of overheating and the speed of the temperature change. The rule set can be now applied to a trigger definition like the following:

```
CREATE TRIGGER OverheatingTrigger
     AFTER UPDATE OF temp ON motor
     WHEN (OverheatingAlarmLevel (motorTemperatures,
motorTempDeltas) > 1.5)     (NotifyTempAlarm@Alarms)
     SEND RULE RESULTS
```

The above trigger is fired on each UPDATE on table motor, and the rules set is evaluated. If the comparison predicate yields true, the action named NotifyTempAlarm() is invoked in the action handler process Alarms. Note that, in the underlaying RapidBase architecture, actions are external and detached [WKP96]. The action requests are satisfied by arbitrary handler programs executing in their own process space.[1] The clause SEND RULE RESULTS causes the defuzzified rule set result to be send, as an action parameter, along with the action request, to the action handler process.

I. CA-fuzzy triggers

So far we have considered C-fuzzy triggers which extend the conventional condition part of regular triggers to include fuzzy predicates and encapsulate fuzzy inference. The next step is to introduce fuzzy actions and to integrate them with the condition part more tightly. Such a trigger, called a *CA-fuzzy trigger*, may be built by embedding action specifications in fuzzy rules. CA-fuzzy triggers provide another dimension to the causality in active databases. Whereas the cause-and-effect between the condition and the action part of a regular trigger (C-fuzzy trigger) is a matter of yes/no, it is a matter of degree in CA-fuzzy triggers.

---

[1] Accordingly, the syntax of the action part of the trigger definition is: <action-name>@<process-name>.

A CA-fuzzy trigger can be symbolically represented as E(CA)*, meaning that an event fires the evaluation of *n* (nÚ1) fuzzy *if-then* rules having a fuzzy antecendent and a fuzzy action concequent.

A. Condition-Action model

In this model, the cause-effect relationship between the database state and the concrete (implemented) actions is expressed as a fuzzy rule set in the form:

if FP_1 *then* FA_1
if FP_2 *then* FA_2

...
if FP_n *then* FA_n

where FP_i is a fuzzy predicate in the form proposed for C-fuzzy triggers and FA_i is a fuzzy action proposition represented as:

Z is z_i

Where *Z* is a linguistic variable such that $T(Z) = \{z_1, z_2, ... z_n\}$. There also exists a set of concrete actions $A = \{a_1, a_2, ... a_n\}$ of which each a_i (i = 1, 2, ... n) may be implemented, in a real system, as a distinct procedure—a database command (or sequence thereof) or an external procedure. The sets *Z* and *A* are said to be *mapped* to each other if z_i is associated with a_i, for each i = 1, 2, ... n..

Thus, a linguistic term z_i denotes a fuzzy action and it is uniquely associated with a concrete action. All the membership functions of *Z* are defined over the same arbitrary domain where the relationships among the fuzzy actions are captured.

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.

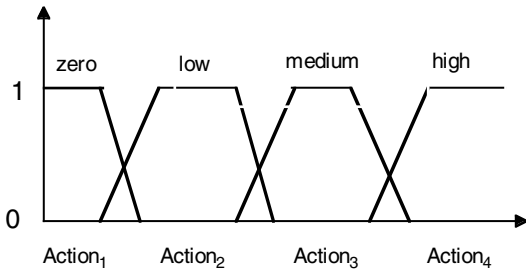

Figure 2. Example of membership functions of fuzzy actions.

The CA-fuzzy execution model is as follows:

(1) When an event is signaled, the interpolative reasoning process (based, for example on the Max-Min algorithm) is performed, and a result Z^* formed by the fusion of the rule results is produced. In Fig. 3, a possible fuzzy result value of Z^*, *low-medium*, is shown if the terms of the preceding figure are used in the fuzzy action proposition.

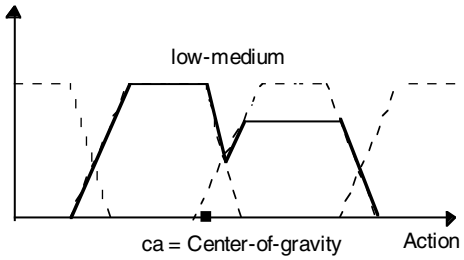

Figure 3. Example of a fuzzy result action.

(2) The fuzzy result is then interpreted by invoking concrete actions, using one of possible strategies, such as:

- **Multiple invocation** maps the fuzzy result action to zero, one or more pre-defined fuzzy actions and invokes the corresponding concrete actions. The strategy is to:

  (1) Defuzzify the result, yielding the crisp action value, *ca.*

  (2) Find fuzzy actions z_i, $i \in \{1,2,...n\}$ such that $\mu_{zi}(ca) \neq 0$, i.e. having non-zero membership values for *ca.*

  (3) Select the corresponding concrete actions for execution.

  For example, in Fig.3, the Center-of-gravity algorithm yields the crisp value *ca* represented by the dot on the Action axis. It is mapped to two fuzzy actions: *low* and *medium* and the corresponding concrete actions $Action_2$ and $Action_3$.

- **Unique invocation** maps the fuzzy result to exactly one fuzzy action. The policy of selecting an action is to

  (1) defuzzify the result, yielding the crisp action value, *ca.*

  (2) Select the fuzzy action whose membership function yields the highest value, for the given *ca*, .i.e.

  z_i *such that* $\mu_{zi}(ca) = \max(\mu_{z1}(ca), \mu_{z2}(ca), ...\mu_{zn}(ca))$.

  For example, in Fig.3,. the crisp value *ca* is mapped to the fuzzy action *low* and, subsequently, to the concrete action $Action_2$.

  (3) Select the corresponding concrete action for execution.

A. Example

Let us consider the example presented in section 3.4 which deals with alarm treatment in a drive control system. We will show that the trigger specification can be further simplified using a CA-fuzzy trigger. Let us assume there are three different alarm notification actions of which at most one is to be invoked.

First, an *action set* is defined whereby terms of the linguistic type AlarmSeverity are mapped to concrete actions:

```
CREATE ACTION SET Alarms OF AlarmSeverity (
     low      NotifyLowAlarm@AlarmServer,
     medium   NotifyMediumAlarm@AlarmServer,
     high          NotifyHighAlarm@AlarmServer)
```

In the trigger definition, a CA rule set is included (following the WHEN keyword). The INPUT clause is used to define the linguistic types and quantifier types of the input variables, and the OUTPUT clause is used to

specify the action set(s). Both for input and output variables, optional alias specifications (following the keywords AS) are possible. Aliases enable to use the most intuitive words in the rule formulation:

```
CREATE FUZZY TRIGGER GeneralOverheatingTrigger
     AFTER UPDATE OF temp ON motor
     INPUT
     motorTemperatures Temperature
          QUANTIFIED WITH Amounts AS motors,
     motorTempDeltas NegativeToPositive
          QUANTIFIED WITH Amounts AS deltas,
     OUTPUT Alarms AS AlarmNotification
WHEN (
     IF some motors ARE hot
          AND most deltas ARE big_positive
          THEN AlarmNotification is low,
     IF some motors ARE very_hot
          AND some deltas ARE big_positive
          THEN AlarmNotification is low,
     IF some motors ARE very_hot
          AND most deltas ARE big_positive
          THEN AlarmNotification is medium,
     IF most motors ARE hot
          AND some deltas ARE big_positive
          THEN AlarmNotification is medium,
     IF most motors ARE hot
          AND most deltas ARE big_positive
          THEN AlarmNotification is high,
     IF most motors ARE very_hot
          THEN AlarmNotification is high)
UNIQUE ACTION
```

The UNIQUE ACTION clause specifies the unique action invocation policy described above. Note also, that the CA-fuzzy trigger above does the job of three C-fuzzy triggers, as one of three different actions may be invoked. Note that, in the CA-fuzzy trigger model, more than one fuzzy action variable may be specified in a single trigger. This expands the capability of a trigger to invoke different concrete actions.

I. Implementation notes

C-fuzzy triggers are being implemented in the prototype active database system TEMPO at VTT Information Technology, Espoo, Finland. TEMPO is an extension of the active time series database system RapidBase [WKP96] which was implemented in the framework of the RAPID project (1992–96). RapidBase is a main-memory-based system using a temporal-relational model and a language based on SQL. The active capabilities include ECA triggers with primitive database events and pre-defined composite event types (e.g., a timer event). The trigger definition syntax is based on the SQL3 [SQL3] proposal. The RapidBase Server is implemented as a multithreaded process programmed in C++ and running in UNIX and Windows NT environments. It is targeted for industrial applications requiring fast access to time-series-formatted data like process measurements.

TEMPO adds the capability to use rule set function calls in comparison predicates, in the condition part of the RapidBase triggers. The *CFlie*[1] fuzzy logic inference engine package is used to implement the rule set functions. The C-fuzzy trigger functionality was derived from a case study [PBW96] where requirements and specific problems of a real industrial application were analyzed, and the syntactical shape of the C-fuzzy trigger was proposed.

---

[1] Originally developed as *Flie* at Institute of Robotics, ETH, Zurich, Switzerland; converted to C at Lab. for Concurrent Computing Systems, Swinburne Univ. of Technology, Hawthorn, Australia.

We are planning to implement other fuzzy trigger models described in the paper later on. More study will be also performed on the needs of the industrial user community in order to achieve the most convenient interfaces of the proposed mechanisms.

I. Conclusions

Various applications, for example in industrial systems, require fuzzy concepts to capture the relevant semantics. To do this, we propose fuzzy database triggers (fuzzy active database rules) which aim to combine two important areas in database technology: fuzzy databases and active databases. In this paper, we first extended the basic semantics of event-condition-action rules with fuzzy predicates and fuzzy inference. Then, more general models were proposed in which different levels of fuzziness were identified, leading to fuzzy actions. By way of examples using concrete syntax, we showed that the proposed models may result in intuitive and user-oriented interfaces.

There remain several issues which require further investigations, such as a general model of fuzzy events capturing event composition. Another important issue is to study the inter-relationship between the proposed fuzzy concepts and other behavioral dimensions, like coupling modes, event consumptions, etc., of active database systems [PDW+93]. We also intend to investigate a formal approach to define the semantics of fuzzy triggers. For example, a formal semantic for fuzzy triggers could be based on the Extended Event-Condition-Action (EECA) rule model [FT95] which captures the semantics of rules from most existing commercial systems and research prototypes. This could be done by extending the ECCA syntax and by providing translation rules of the new features into a logical style of rule format, called the*core* format.

References

[BD83] B.G. Buchanan and R.O. Duda. Principles of Rules-Based Expert Systems". In Advances in Computers, 1983, Vol. 22, pp. 163-216.

[BH95] Mikael Berndtsson, Jörgen Hansson (Eds.): Active and Real-Time Database Systems (ARTDB-95), Proceedings of the First International Workshop on Active and Real-Time Database Systems, Skövde, Sweden, 9-11 June 1995.

[BP93] P. Bosc, H. Prade: An introduction to the fuzzy set and possibility theory-based treatment of soft queries and uncertain or imprecise databases. In IRIT Research report 93-57-R, Toulouse, France, December 93, 27 pages.

[BP95] P. Bosc, O. Pivert "SQLf: A relational database language for fuzzy querying". In IEEE Transactions On fuzzy Systems, Vol. 3, NO. 1, Feb. 1995, pp.1-17.

[CM94] Sharma Chakravarthy, D. Mishra "Snoop: An Expressive Event Specification Language for Active Databases". In Data & Knowledge Engineering , Volume 14, 1994, pp. 1-26.

[DHW94] Dayal, U. & Hanson, E.N. and Widom, J. "Active Database Systems." In W. Kim editor, Modern Database Systems: The Object Model, Interoperability and Beyond, Addison-Wesley, Reading, Massachusetts, September 1994, pp. 434-456.

[DEB89] Data Engineering Bulletin-Special Issue on Imprecision in Databases, Volume 12, Number 2, June 1989.

[DP88] Didier Dubois, Henri Prade. "Possibility theory: An approach to computerized Processing of Uncertainty". Plenum Press, New York, 1988.

[FT95] Piero Fraternali, Letizia Tanca: A Structured Approach for the Definition of the Semantics of Active Databases. In ACM Transactions on Database Systems (TODS), 20(4), 1995, pp. 414-471.

[GD93] Gatziu, S. and Dittrich, K.R. Events in an Active Object-Oriented Database System. In Proceedings of the 1st International Workshop on Rules in Database Systems (RIDS) , Edinburg (Scotland), August 1993, pp. 23-39.

[GKB+84] Gupta M.M, Kandel, A., Bandler, W. and Kiszka, J. (Eds.). "Approximate reasoning in expert systems". Elsevier Science Publishers, North-Holland, 1984.

[GJS92] Gehani, N.H. & Jagadish, H.V. and Shmueli, O. Event Specification in an Active Object-Oriented database. In ACM SIGMOD Conference on Management of Data, San Diego, California, June 1992, pp. 81-90.

[IS89] Information Systems, Volume 14, Number 6, 1989.

[LWL89] K. S. Leung, Man Hon Wong, W. Lam. "A Fuzzy Expert Database System". In Data & Knowledge Engineering , Volume 4, 1989, pp. 287-304.

[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.

[PDW+93] Paton, N.W. & Diaz, O. & Williams, M.H. & Campin, J. & Dinn, A. and Jaim, A. "Dimension of Active Behavior". In Proceedings of the 1st Int. Workshop on Rules in Database Systems, Edinburg (Scotland), August 1993, pp. 40-57.

[Pet96] Frederick E. Petry. "Fuzzy Databases: Principles and applications". With contribution by Patrick Bosc, International Series in Intelligent Technologies, 1996, 240 pages.

[PBW96] Pesonen, A., Bouaziz, T., and Wolski, A. Case Study: Applying Fuzzy Triggers to a Drive Control System. Research Report No. J-6/96, VTT Information Technology, Espoo, Finland, August 1996.

[RSS+96] Krithi Ramamritham, Rajendran M. Sivasankaran, John A. Stankovic, Donald F. Towsley, Ming Xiong "Integrating Temporal, Real-Time, and Active Databases." In SIGMOD Record 25(1), 1996, pp. 8-12.

[SQL3] Working Draft Database Language SQL3, J. Melton (ed.), August 1994, ANSI X3H2-94-329, ISO DBL:RIO-004.

[VK91] Van der Voort, M.H. and Kerstein, M.L. "Facets of Database Triggers". Amsterdam : CWI, April 1991, Technical Report N° CS-R9122, 35 pages.

[VSC+89] R. Vandenberghe, A. Van Schooten, R. De Caluwe, E. E. Kerre: Some practical aspects of fuzzy database techniques: an example. In [IS89], pp. 465-472.

[WC96a] Jennifer Widom, Stefano Ceri (Eds.). "Active Database Systems: Triggers and Rules For Advanced Database Processing". Morgan Kaufmann, 1996.

[WC96b] Jennifer Widom, Stefano Ceri. "Introduction to Active Database Systems". In [WC96a], pp. 1-41.

[WKP96] Wolski A., Karvonen J., Puolakka A. 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 ftp:://ftp.vtt.fi/pub/projects/rapid/case.ps).

[Zad65] Lofti A. Zadeh: Fuzzy Sets. In Information Control, Vol. 8(3), June 1965, pp. 338-353.

[Zad78] Lofti A. Zadeh. "Fuzzy sets as a basis for a theory of possibility". In Fuzzy sets and Systems, Vol. 1, 1978, pp. 3-28.

[Zad84] Lofti A. Zadeh. "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems". In [GKB+84], pp. 3-31.

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

[ZK84] Maria Zemankova and A. Kandel: Uncertainty propagation to Expert Systems. In [GKB+84], pp. 529-548.

[Zem89] Maria Zemankova. FILIP: a fuzzy intelligent information system with learning capabilities. In [IS89], pp. 473-486.