FORMAL METHOD TO IMPLEMENT FUZZY REQUIREMENTS

MÉTODO FORMAL PARA IMPLEMENTAR REQUERIMIENTOS DIFUSOS

MARLENE GONCALVES
Ph.D, Universidad Simón Bolívar, Caracas, Venezuela, mgoncalves@usb.ve

ROSSELINE RODRÍGUEZ
Ph.D, Universidad Simón Bolívar, Caracas, Venezuela, crodrig@usb.ve

LEONID TINEO
Ph.D, Universidad Simón Bolívar, Caracas, Venezuela, leonid@usb.ve

Recibido para revisar 15 de abril de 2011, aceptado Diciembre 21 de 2011, versión final 15 de Enero de 2012

ABSTRACT: Many user requirements may involve preference criteria linguistically expressed by fuzzy terms in natural language; these requirements are called fuzzy requirements. Database query languages have been extended incorporating fuzzy logic to handle user-preference criteria. To the best of our knowledge, very few of the software development methods consider fuzzy queries. In this work, we propose a database application method which includes conversion rules that translate formal specifications to implementations in the structured query language (SQL) enhanced with fuzzy logic (SQLf). The novelty of our method is the tuple calculus extension in order to express fuzzy queries with formal specification. Also, our method includes conversion rules that translate formal specifications into implementations in SQLf, a fuzzy query language on crisp databases. Additionally, we illustrate how our method was successfully applied in a real case study.

KEYWORDS: formal specifications, fuzzy queries, fuzzy terms, relational calculus, software development methodology, SQLf.

RESUMEN: Muchos requerimientos de usuario pueden involucrar criterios de preferencia expresados en el lenguaje natural por medio de términos difusos; éstos son llamados requerimientos difusos. Por otro lado, los lenguajes de consulta a bases de datos han sido extendidos incorporando la lógica difusa para manejar las preferencias de usuarios. Pocas de las metodologías conocidas para el desarrollo de aplicaciones sobre base de datos consideran las consultas difusas. En este trabajo, se propone un método para aplicaciones a bases de datos cuyo objetivo es desarrollar sistemas de software con soporte de consultas difusas. Lo novedoso de éste es la extensión al cálculo de tuplas para la especificación formal de consultas difusas. Además, el método incluye reglas de traducción de una especificación formal a una consulta en SQLf (structured query language + fuzzy logic), un lenguaje de consultas difusas sobre bases de datos precisas. Se ilustra su utilidad con la aplicación a un caso de estudio real.

PALABRAS CLAVE: especificaciones formales, consultas difusas, términos difusos, cálculo relacional, metodología de desarrollo de software, SQLf.

1. INTRODUCTION

Traditional applications retrieve data from database systems applying Boolean condition filters. Nevertheless, user requirements may involve fuzzy terms that represent user's preferences over data. Fuzzy terms correspond to concepts whose boundaries are not defined clearly and/or whose semantics are susceptible to domain and/or user preferences. These concepts may be modeled using fuzzy sets [1] that allow for the gradual membership of elements. In this paper, requirements comprised of these concepts are named *fuzzy* requirements. For example, let us suppose someone is interested in knowing how *easy* the courses are. The *easy* adjective is a fuzzy term. Easiness depends on user-preferences. For example, someone can define an *easy* course if all students receive an A-grade, but another person can be more flexible defining it as most of students get a grade greater or equal than B. In addition, someone may define an *easy* course as one in which all students have a *high* grade. Notice that *high* is also a fuzzy term.

In order to provide fuzzy requirement support, several extensions of structured query language (SQL) have been proposed, such as SQLf [2], FSQL [3], and

SoftSQL [4]. Among these proposals, SQLf is the most complete because it is updated with SQL2003 features [5]; therefore, we would like our research to be compatible with SQLf.

Despite advances in software engineering and the existence of sophisticated computer-aided software engineering (CASE) tools, automatic code generation from models is still something hoped for [6]. Our final goal is to provide automated software engineering tools for developing applications with fuzzy requirements. In this sense, the main activities of software development models have been extended in order to support fuzzy requirements [7]. The authors in [8] proposed a method based on object constraint language (OCL) and fuzzy logic for the development of applications with fuzzy requirements. We need to include fuzzy features in a formal language such as tuple [9] or domain [10] relational calculus. Thus, since natural language may be ambiguous, requirements must be specified in a formal language for guaranteeing system correctness.

In [11], Galindo et al., have presented a definition of fuzzy domain relational calculus framed in the GEFRED model. In [12] a domain calculus was proposed for Buckles-Petry's fuzzy relational database model that, according to Galindo et al. [11], is much more restrictive. Their research has restricted logic expressions to the use of classic universal and existential quantifiers without considering more general fuzzy ones. The fact that they have been based on domain calculus is not compatible with SQLf. The use of domain calculus results in satisfaction degrees for fuzzy conditions on attribute values, but SQLf computes degrees of satisfaction for the whole tuple. The domain calculus extension is suitable for handling fuzzy attribute values as is FSQL. In [13] the authors proposed a fuzzy query language in terms of relational calculus, but their work is based on domain calculus [10]. Quantified propositions in [13] satisfy Zadeh's interpretation that has some disadvantages and is not adequate for data base queries [14]. Authors in [15] extend tuple calculus with fuzzy logic. Their work focuses on a language for preference expressions in route planning. Nevertheless, we need a generic language for the formal expression of fuzzy requirements with the possibility of formal proofs and a mechanism to translate requirements to an implementation language (SQLf).

We propose a method based on a formal specification which allows for developing applications with the fuzzy requirement. Formal specifications are done with an extension of tuple calculus that incorporates fuzzy conditions as the novel contribution of this work. Since formal specifications in tuple calculus are symbolic logic expressions, they allow for one to perform formal tests in order to verify the correctness of the requirements. We do not address all methodological aspects of the system such as user interfaces or correctness of requirements for data insertion, reports, interaction with other systems, etc. We just focus on fuzzy requirements.

The paper is comprised of five sections including the introduction. In Section 2, we briefly describe background on fuzzy sets and SQLf. In Section 3, we explain our development method for database applications which are characterized by fuzzy requirements. This method includes an extension of tuple calculus to formally specify fuzzy queries, and translation rules to implement those fuzzy queries like SQLf statements. In Section 4, we apply our method in a real case study. Lastly, in Section 5, the concluding remarks and suggestions for future work are given.

2. BACKGROUND

Fuzzy sets [1] are defined by means of membership functions from a base universe or domain to the real interval [0,1]. The set of elements whose membership degree is greater than zero is the *support*. The *core* is the set of elements whose membership degree is equal to one. The *border* is the set of elements whose membership degree is neither zero nor one. Fuzzy set theory is the basis of fuzzy logic, where truth values are in [0,1]; the zero value represents "completely false", and one value is "completely true". The truth value of a proposition "s" is denoted by μ(s). Fuzzy logic may be used to specify searching conditions in query, according to an SQL extension named SQLf [2][5].

A query can be expressed as follows:

```
SELECT <columns>
FROM <tables>
WHERE <fuzzy conditions>
WITH CALIBRATION [k|α|k,α]
```

The `SELECT` clause projects columns that are retrieved from the tables specified in the `FROM` clause, and the `WHERE` clause specifies a fuzzy condition that must be satisfied by retrieved rows. The `WITH CALIBRATION` clause is optional and allows retrieving: a) the `k` best answers, and/or b) rows whose membership degree is greater or equal than α value.

Tineo [17] introduced SQLf-DDL, a data definition language for fuzzy terms, according to the Zadeh [16] classification. It allows for the specification of fuzzy predicates, modifiers, comparators, connectors, and quantifiers. The definition syntax varies depending on the kind of term, but generally it follows the structure:

CREATE FUZZY <kind>

<name> [ON <dom>] AS <def>

Where: `<kind>` is one of the keywords `PREDICATE`, `MODIFIER`, `COMPARATOR`, `CONNECTOR` or `QUANTIFIER`; `<name>` is an identifier denoting the fuzzy term; and `<def>` is a complex expression defining the operational semantics of the fuzzy term. The `ON <dom>` clause is only for predicates. It is intended to specify the base universe or domain of the fuzzy set.

3. OUR METHOD

SQLf has been used for some developments [18]. Based on these experiences, we propose a method for developing applications that support fuzzy requirements. Firstly, the analysis produces a list of fuzzy requirements in natural language, where fuzzy terms are normally used. Linguistic terms of a vague nature are identified and represented using fuzzy theory. Analysts determine which fuzzy terms are necessary, their types, and their definitions. Secondly, each fuzzy requirement is written in tuple calculus using user-defined fuzzy terms. Thirdly, the software system may be built using SQLf. In this step, fuzzy requirement specifications in tuple calculus are translated into SQLf.

3.1. Fuzzy requirement analysis

From a user's requirements in natural language, we may determine grammatical elements such as adjectives and adverbs which are indicators of vagueness. Qualifying adjectives refer to quality and have several levels of intensity: The *positive* level corresponds to an adjective in its original form such as: *good, bad, cheap,* and *expensive*. Usually, they are represented as fuzzy predicates. The *comparative* level of an adjective is expressed in English by "–er" or "more" (e.g., *cheaper* and *more expensive*). Also, there are pure adjectives such as *better* or *worse* which are fuzzy comparators. The *superlative* form of an adjective is expressed in English by "–est"; e.g., "most" or "least", as in *the most efficient* or *the youngest*. Other superlatives are the following words: *optimal, supreme,* or *extreme*. A "superlative" degree indicates a comparison between elements of the same set. Superlative representation may require fuzzy modifiers, fuzzy predicates, fuzzy comparators, and/or fuzzy quantifiers. There are also determinative adjectives that are related to quantities, such as *few, many, much,* and *several*. These correspond to fuzzy quantifiers. Adverbs, such as *very* and *extremely* are words that modify a verb or adjective. They may be modeled as fuzzy modifiers.

3.2. Formal specification of fuzzy queries

A formal specification describes behavior and properties of a system written in a formal language. Formal models allow for verification if system descriptions are consistent. Tuple calculus is a formal language used to represent users' requirements over relational databases. A tuple calculus query is an expression in first order logic that identifies its resulting tuples set. We extend tuple calculus with fuzzy logic with a notation for expressions similar to that of [19].

A query in tuple fuzzy calculus is an expression of form: $C = \{t_1.a_1,\ldots,t_n.a_n \mid R(t_1,\ldots,t_n) : P(t_1,\ldots,t_n)\}$. Here $t_1,\ldots,t_n$ are free variables that represent tuples; each a_i is a valid attribute of tuple t_i, or the special attribute specification symbol * whose meaning is the list of all attributes forming the tuple t_i; $R(t_1,\ldots,t_n)$ is a conjunction of expressions with form $t_i \in T_i$ establishing the valid range of tuples. Each T_i is a database table; $P(t_1,\ldots,t_n)$ is a valid formula that states a fuzzy condition to be satisfied by returned tuples. The result of C is a fuzzy set of tuples. There is one tuple for each possible assignation of variables $t_1,\ldots,t_n$ satisfying the range restriction $R(t_1,\ldots,t_n)$ and the fuzzy condition $P(t_1,\ldots,t_n)$. The membership degree of resulting tuples is given by the effective truth value of $P(t_1,\ldots,t_n)$ for the corresponding assignation of variables $t_1,\ldots,t_n$. We denote the effective truth value of a valid formula F as $\mu(F)$.

For convenience, we also allow for a query in tuple fuzzy calculus be an expression of form: $C = \{t_1.a_1,...,t_n.a_n \mid R(t_1,...,t_n) \wedge F(t_1,...,t_n) : P(t_1,...,t_n)\}$ that is defined as equivalent to $\{t_1.a_1,...,t_n.a_n \mid R(t_1,...,t_n): P(t_1,...,t_n) \wedge F(t_1,...,t_n)\}$, where $F(t_1,...,t_n)$ is a valid formula.

3.2.1. Atomic valid formulas

Valid formulas are built on atoms. Classic (crisp) atoms are expressions of form: $t \in T$, with t as a tuple variable and T a database table; or $e\ \theta\ e'$, being e and e', arithmetic expressions built on tuple's attributes and constants, and θ a comparison operator $\theta \in \{=, \neq, <, \leq, >, \geq\}$. The effective truth value of a crisp atom would be 1 for true or 0 for false. Fuzzy atoms contain fuzzy terms: predicates, modifiers, or comparators.

The expression *t.a is fp* is an atom where *t* is a tuple variable, *a* is an attribute of *t*, *fp* is a fuzzy predicate, and the "*is*" keyword is an operator for fuzzy predicates on linguistic variables. For example, a fuzzy expression for a high grade point average (gpa) could be "*t.gpa is high*", where *high* is a linguistic label that corresponds to a fuzzy predicate. An interpretation for a predicate *fp* is a fuzzy set whose membership function is denoted as μ_{fp}. Thus, given an assignation where the attribute *t.a* takes the value *v*, the logical expressions "*t.a is fp*" effective truth value would be $\mu(t.a\ is\ fp) = \mu_{fp}(v)$.

Fuzzy comparators are expressed as binary operators between crisp values: $e_1\ fc\ e_2$ where *fc* is a linguistic label for the fuzzy comparator, e_1 and e_2 are arithmetic expressions built on tuple's attributes and constants. For instance, "t_1.*gpa worseThan* t_2.*gpa*" is an expression using the comparator "*worseThan*". An interpretation for a fuzzy comparator *fc* is a fuzzy set of pairs whose membership function is denoted as μ_{fc}. Thus, given an assignation where the expressions e_1 and e_2 take the value v_1 and v_2, respectively; $\mu(e_1\ fc\ e_2)$, would be $\mu_{fc}(<v_1,v_2>)$.

The expression *t.a is fm fp* is an atomic valid formula where *fm* is a fuzzy modifier, *fp* is a fuzzy predicate, *t* is a tuple variable, and *a* is an attribute of *t*. For example, *t.gpa is very high*. An interpretation for the fuzzy modifier *fm* is a transformation over fuzzy sets' membership functions. Given μ_{fp}, a membership function, the fuzzy modifier *fm* produces a transformed membership function $\mu_{fm(fp)}$. Thus, given an assignation where the attribute *t.a* takes the value *v*, $\mu(t.a\ is\ fm\ fp) = \mu_{fm(fp)}(v)$.

3.2.2. Combined valid formulas

Valid formulas allow parenthesis use; e.g., *(F)* is equivalent to *F*. If *F* is a valid formula, then $\neg F$ is a valid formula with $\mu(\neg F) = 1\text{-}\mu(F)$. If F_1 and F_2 are valid formulas then $F_1 \wedge F_2$, $F_1 \vee F_2$ and $F_1 \Rightarrow F_2$ are valid formulas, with semantics: $\mu(F_1 \wedge F_2) = min(\mu(F_1),\mu(F_2))$, $\mu(F_1 \vee F_2) = max(\mu(F_1),\mu(F_2))$ and $\mu(F_1 \Rightarrow F_2) = max(\mu(\neg F_1), \mu(F_2))$. Valid formulas may be also combined using fuzzy connectors. Let "*fn*" be a fuzzy connector: We will use prefix notation for unary connectors as "*fn F*". For binary connectors the notation would be infixed as "$F_1\ fn\ F_2$". For example, suppose "*fimp*" is a linguistic label for a fuzzy implication, we may use this connector in the expression (*t.gpa is highest*) *fimp* (*t.gpa is high*). An interpretation for the unary fuzzy connector *fn* is a unary closed operator σ_{fn} on the real interval [0,1]. Thus, $\mu(fn\ F)$ would be $\sigma_{fn}(\mu(F))$. In the same way, for a binary fuzzy connector *fn*, an interpretation would be a binary closed operator σ_{fn} in [0,1]. Thus, $\mu(F_1\ fn\ F_2)$ would be $\sigma_{fn}(\mu(F_1),\mu(F_2))$.

3.2.3. Quantified valid formulas

Fuzzy quantifiers represent imprecise quantities as an extension of existential and universal quantifiers. They allow for the building of valid formulas with the following notation based on [19]: $(fq\ x: R(x): P(x))$ where *fq* is a linguistic label for a fuzzy quantifier, *x* is a variable linked to quantifier, *R(x)* is the variable range and *P(x)* is a valid formula. *R(x)* may be either of the form $x \in X$ or the form $x \in X \wedge F(x)$, being *F(x)* a valid formula. For example, the statement "most of the students have high gpa" may be expressed as (*mostOf e : e ∈ Student: e.gpa is high*).

Interpretation of a fuzzy quantifier *fq* may be given by fuzzy sets of numbers with a non-empty core and a convex membership function μ_{fq}. Several measures have been proposed for the effective truth value of a fuzzy quantified formula; we assume Tineo's [20].

3.3. Formal specification translation

Requirements may be implemented in SQLf. Each fuzzy term is defined using SQLf-DDL [17]. In addition, queries expressed in tuple calculus are translated into SQLf. The following translation rules are an extension of those for classic queries presented in [21].

Translation of an $C = \{t_1.a_1, ..., t_n.a_n \mid R(t_1, ..., t_n) : P(t_1, ..., t_n)\}$ expression is as follows:

- The attributes $t_1.a_1, ..., t_n.a_n$ are included in the `SELECT` clause.
- Tables T_i in range of membership $R(t_1, ..., t_n)$ are specified in the `FROM` clause incorporating in the `AS` clause the corresponding range variable as follows: "`FROM` T_1 `AS` $t_1, ..., T_n$ `AS` t_n".
- Condition $P(t_1, ..., t_n)$ is expressed into the `WHERE` clause. It is necessary to normalize the formula of P in order to translate this condition. Since SQLf is an extension of SQL, it does not have a universal quantifier but an existential one (`EXISTS`). However, this extension allows for representing any fuzzy quantifier. The normalization procedure consists of four steps [21]: First, eliminate all implications by applying the implication equivalence: $(F_1 \Rightarrow F_2) \equiv (\neg F_1 \vee F_2)$; second, eliminate all universal quantifiers by applying the universal quantification equivalence: $(\forall t{:}R(t){:}F(t)) \equiv \neg(\exists t{:}R(t){:}\neg F(t))$; third, eliminate double negation by applying the double negation equivalence: $\neg\neg F \equiv F$; fourth, eliminate sandwiched negation by applying De Morgan's equivalence: $\neg(F_1 \wedge F_2) \equiv (\neg F_1 \vee \neg F_2)$ and $\neg(F_1 \vee F_2) \equiv (\neg F_1 \wedge \neg F_2)$. The highlighted negation in the formula $\neg(\exists t{:} R(t){:}\boldsymbol{\neg(F(t))})$ is referred by a sandwiched negation. For example, the expression $(\forall t{:}\ R(t){:}\ t>0 \wedge t<100) \equiv \neg(\exists t{:}\ R(t){:}\ \neg(t>0 \wedge t<100))$ applying universal quantification equivalence where $\neg(t>0 \wedge t<100)$ is a sandwiched negation; the expression $\neg(\exists t{:}\ R(t){:}\ \neg(t>0 \wedge t<100)) \equiv \neg(\exists t{:}\ R(t){:}\ \neg(t>0) \vee \neg(t<100))$ applying De Morgan's equivalence.

If all formulas of a query $C = \{t_1.a_1, ..., t_n.a_n \mid R(t_1, ... t_n) : P(t_1, ... t_n)\}$ are normalized, we define the translation function (Trans) as:

Trans(F) = F if F is an atom different from $t_i \in T_i$

Trans($F_1 \wedge F_2$) = Trans(F_1) `AND` Trans(F_2)

Trans($F_1 \vee F_2$) = Trans(F_1) `OR` Trans(F_2)

Trans(F_1 fn F_2) = Trans(F_1) fn Trans(F_2)

Trans((F)) = Trans(F)

Trans($\neg F$) = `NOT` Trans(F)

Trans(fn F) = fn Trans(F)

Trans($\exists t_1, ..., t_n : R(t_1, ..., t_n){:}\ F(t_1, ..., t_n)$) = `EXIST` (`SELECT` $t_1, ..., t_n$ `FROM` T_1 `AS` $t_1, ..., T_n$ `AS` t_n `WHERE` Trans($F(t_1, ..., t_n)$))

A query $C = \{t_1.a_1, ..., t_n.a_n \mid R(t_1, ... t_n) : P(t_1, ... t_n)\}$ without fuzzy quantifiers, is translated as: Trans(C) = `SELECT` $t_1.a_1, ..., t_n.a_n$ `FROM` T_1 `AS` $t_1, ..., T_n$ `AS` t_n `WHERE` Trans($P(t_1, ..., t_n)$) ;

A query $C = \{t_1.a_1, ..., t_n.a_n \mid R(t_1, ..., t_n) \wedge F_1(t_1, ..., t_n) : (fq\ s : s \in S \wedge F_2(s) : F_3(t_1, ..., t_n, s))\}$ where F_1, F_2, and F_3 do not contain fuzzy quantifiers, is translated as: Trans(C) = `SELECT` $t_1.a_1, ..., t_n.a_n$ `FROM` T_1 `AS` t_1, ... ,T_n `AS` t_n, S `AS` s `WHERE` Trans($F_1(t_1, ..., t_n)$) `GROUP BY` $t_1.k_1, ..., t_n.k_n$ `HAVING` fq Trans($F_2(s)$) `ARE` Trans($F_3(t_1, ..., t_n, s)$); Where $k_1, ..., k_n$ are key attributes of $T_1, ..., T_n$. When $F_1(t_1, ..., t_n)$ is absent in query C, we omit `WHERE` Trans($F_1(t_1, ..., t_n)$) in translation. When $F_2(s)$ is absent in query C, we omit Trans($F_2(s)$).

4. OUR CASE STUDY

To illustrate the application of our method, we developed a database application for managing a student opinion survey (SOS) at the Simon Bolivar University (in Spanish, *Universidad Simón Bolívar*, USB). The SOS instrument consisted of 31 items and their answers were integers between 1 and 5. The items corresponded to "Opinion about Professor's performance", "General Opinion", "Student Self-Evaluation", and "Opinion about the course". The Simon Bolivar University has an SOS database that stores student opinions about courses and professors for several trimesters. SOS database size is about 4,000,000 tuples. All this information may be useful for decision making and teaching quality improvement. The objective is to support decision making in our university community.

4.1. Fuzzy requirement analysis

We identify different actors for the SOS system with different privileges and requirements. The Office of Information Engineering at USB controls privileges. Our focus in this paper is only in fuzzy requirements. Using natural language, we specify some of the identified user's requirements in Table **1**. We identify fuzzy terms in the user's requirements and we emphasize them in italics, bold, and in parentheses.

Table 1. Some requirements in natural language

ID	Requirement description
C1	For a department d and a period p, in which courses most of students might obtain a (low, regular, or high) grade?
C2	For a professor o and a period p, which are o's (weak or outstanding) issues according most students' opinion?
C3	For a department d and a period p, how (easy, regular, or difficult) are the courses, in terms of difficulty level, available resources, correspondence to number of credits, and grade expected for course c?

The following step is for defining each identified fuzzy term. These terms will be used in the formal specifications of the fuzzy requirements. Terms *low*, *regular,* and *high* are positive adjectives and they are modeled as fuzzy predicates. Figure 1 shows trapezoidal representations for them.

We will model *outstanding* by means of the linguistic expression *very high* where *very* is a fuzzy modifier that translates the membership function of fuzzy predicate by 1 unit in the abscissas axis. If the *high* predicate was defined as in Fig. 1, the *very* modifier will be represented as in Fig. 2.

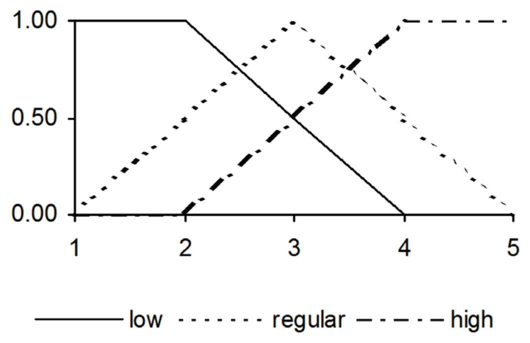

Figure 1. Membership function for low, regular, and high predicates in the 1–5 range

The *weak* predicate will be modeled as the negation of the *outstanding* predicate. Since negation is a fuzzy connector whose definition may be interpreted as a complement, we represent it using the membership function of Fig. 2.

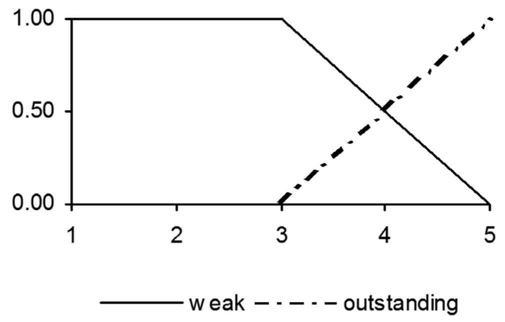

Figure 2. Membership functions representing the terms *outstanding* (defined as *very high*) and *weak* (defined *not outstanding*), being *very* and *not* fuzzy modifiers

The terms *easy*, *regular*, and *difficult* are positive adjectives. When we analyzed SOS, we observed that a set of items determines how *easy*, *regular*, or *difficult* the courses are. Therefore, we need a composed predicate. For example, a course is *easy* when it satisfies three criteria: student's previous preparation is *high*, the dedication required to pass it is *low*, and the availability of support materials is *high*. According to user preferences, the predicates *easy*, *regular*, and *difficult* are defined in Table 2.

Table 2. Definitions of the linguistic terms *easy*, *regular*, and *difficult*

Term	Definition (fuzzy condition)
easy	Previous Preparation = high $\wedge$ Required Dedication = low $\wedge$ Material Availability = high
regular	Previous Preparation = regular $\wedge$ Required Dedication = regular $\wedge$ Material Availability = regular
difficult	Previous Preparation = high $\wedge$ Required Dedication = high $\wedge$ Material Availability = low

Three requirements contain the term *most of*. This term corresponds to a determinative adjective that may be defined as a proportional quantifier because it describes a relative quantity of elements. In Figure 3 its representation is given.

4.2. Formal specification of fuzzy queries

The Office of Information Engineering at USB carried out SOS database design and it comprises previous research to our development. Figure 4 contains a simplification of relational SOS schema. Some tables and attributes have been omitted because they are irrelevant in our requirements. All fuzzy terms have been defined.

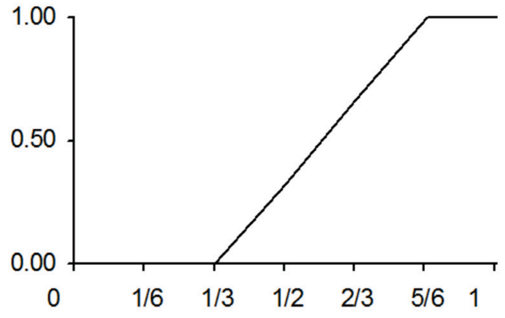

Figure 3. Membership function of fuzzy set defining the linguistic quantifier *mostOf*

Thus, we may specify natural language requirements in relational calculus. To illustrate specification in relational calculus, we only consider some fuzzy requirements.

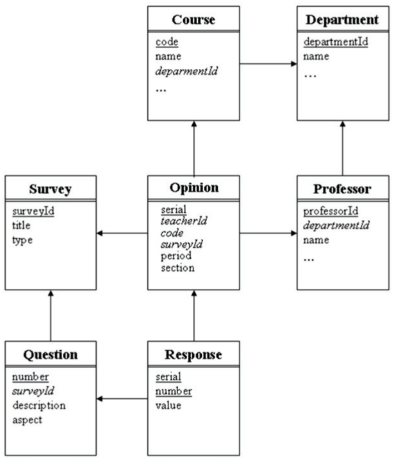

Figure 4. SOS database relational schema. Primary keys are underlined. Foreign keys coincide in name with corresponding referenced keys in the sense of the arrow. Irrelevant attributes are omitted (…).

Suppose the requirement C1: "For a department *d* and a period *p*, which are the courses where *most* students may obtain a *high* grade?" In relational calculus, a query result is represented as a set of tuples that satisfy a condition given by the user. Thus, the user is interested in those courses $(c \in Course)$ where *most* student opinions $(o \in Opinion)$ have a *high* value for linguistic variable "Expected Grade" corresponding to item 25 of the survey. Its formal specification is as follows: $\{c.* \mid c \in Course \wedge c.departmentId = d : (mostOf\ o: o \in Opinion \wedge o.code = c.code \wedge o.period = p : (\forall r: r \in Response: r.serial = o.serial \wedge r.number = 25 \wedge r.value\ is\ high))\}$.

Let us consider the C2requirement: "For a professor *d* and a period ***p***, which are *d*'s *weak* issues according *most* student opinions?" For this requirement, the items from 1 to 19 of the survey corresponding to professor performance are necessary. Also, there is a precondition for professor *d*: $(d \in Professor)$. A description of each issue is represented as an attribute "issue" in table "Question" of Figure 4; e.g., item 1 of SOS has a value of attribute "number" equal 1 and the "issue" attribute value is "clarity of course program". Range of the fuzzy quantifier *mostOf* just involves opinions about professor *d* $(o \in Opinion \wedge o.professorId = d.professorId)$. The "Opinion" table contains information on several surveys filled out by the students and each answer of the survey is in the "Response" table. In consequence, use of the existential quantifier and/or the universal one is equivalent to the specification of this requirement. We use the universal quantifier, and we represent the fuzzy term *weak* by the compound predicate *not very high*. We may formally specify C2 as:

$\{q.issue \mid q \in Question \wedge q.number \in \{1,..,19\} : (mostOf\ o: o \in Opinion \wedge o.professorId = d.professorId \wedge o.period = p : (\forall r : r \in Response : r.serial = o.serial \wedge r.number = q.number \wedge r.value\ is\ not\ very\ high))\}$.

The requirement C3 states: "For a department *d* and a period *p*, how *easy* are the courses?" This is a potentiality example of fuzzy quantifiers over classical ones. In this requirement, the user wants to know a satisfaction degree, which may be directly modeled using the fuzzy quantifier *mostOf*. The result of this quantifier produces an answer in the closed interval [0,1] according to its definition in Figure 3. In quantifier range, we must verify opinions belong to department *d* $(\exists m{:}m \in Course{:}o.code = m.code \wedge m.departmentId = d)$. Notice that if there are no opinions about courses of a department, the result should be 1 because the existential result is false and the range of the fuzzy quantifier is empty or false.

Remember that the ***easy*** predicate is a predicate combined by previous preparation, required dedication, and material availability according to

Table 2. This is survey items 22, 23, and 24, respectively. This corresponds to attribute "*number*" of "Response" table; i.e., *(r.number∈{22,23,24})*. Similarly, we have a precondition: *d* is a department code. Finally, we express this requirement as: *{c.* | c∈ Course ∧ c.departmentId = d : (mostOf o : o ∈ Opinion ∧ (∃m: m ∈ Course : o.code = m.code ∧ m.departmentId = d) ∧ o.period = p : (∀r: r ∈ Response : r.serial = o.serial ∧ r.number ∈ {22,23,24} ∧ (r.number = 22 ⇒ r.value is high) ∧ (r.number = 23 ⇒ r.value is low) ∧ (r.number=24 ⇒ r.value is high)))}*. Notice that each answer of a student opinion corresponds to a different tuple in the "Response" table. In consequence, a condition conjunction over the items 22, 23, and 24 requires a universal quantifier.

4.3. Formal specification translation

In this phase, fuzzy terms and relational calculus queries are translated into SQLf statements. We define the following fuzzy terms in SQLf as:

CREATE FUZZY PREDICATE *low* ON 1..5 AS (INFINITE,INFINITE,2,4);

CREATE FUZZY PREDICATE *regular* ON 1..5 AS (1,3,3,5);

CREATE FUZZY PREDICATE *high* ON 1..5 AS (2,4,INFINITE,INFINITE);

CREATE FUZZY MODIFIER *very*

AS TRANSLATION 1;

CREATE FUZZY QUANTIFIER *mostOf* AS(0.3333,0.8333,1.0,INFINITE);

Note that terms such as *low, regular*, and *high* are defined in terms of the problem context. For simplicity, we use several labels for different contexts. Terms *weak*, *outstanding*, *easy*, *regular,* and *difficult* are represented based on previous user-defined terms. Since there are a high number of courses and professors, all queries assume just one selected department. If the user is a professor, the department selected corresponds to his department. In queries that require a specific professor or a specific course, the user must select from a list specific to his department. In those queries containing a comparison, the user must previously choose an academic period.

We will describe implementation of fuzzy queries specified formally in the previous section. We assume the user has selected a department, which is in the variable *$sel1*. Also, the professor and period selected correspond to variables *$sel2* and *$sel3*. Threshold is specified by the user in the variable *$sel4*. It would be used in the calibration clause.

The formal specification of requirement C1 must be normalized. First, we apply the universal quantification equivalence; and second, we apply De Morgan's equivalence:

C1 = {c.* | c ∈ Course ∧ c.departmentId = $sel1 : (mostOf o: o ∈ Opinion ∧ o.code = c.code ∧ o.period = $sel3 : ¬(∃r: r ∈ Response: r.serial ≠ o.serial ∨ r.number≠25 ∨ r.value is not high))}.

Using the normalized formula of *C1*, we may apply the function Trans(*C1*) in order to specify the corresponding SQLf query: `SELECT c.* FROM Course AS c, Opinión AS o WHERE c.departmentId = $sel1 GROUP BY c.code HAVING mostOf o.code = c.code AND o.period = $sel3 ARE NOT EXIST (SELECT r FROM Response AS r WHERE r.serial <> o.serial OR r.number <> 25 OR r.value is not high) WITH CALIBRATION $sel4;`

`GROUP BY` clause partitionates surveys by courses. The `mostOf` quantifier indicates proportion of tuples in each partition satisfying the fuzzy condition "`r.value is high`"; i.e., most surveys say that a high grade is expected.

The normalized requirement *C2* is

C2 = {q.issue | q Question q.number {1,..,19} : (mostOf o: o Opinion o.professorId = $sel2 o.period=$sel3 : (∃r: r Response : (r.serial o.serial r.number q.number r.value is very high)))}.

Based on the function Trans(*C2*), the SQLf query is as follows: `SELECT q.issue FROM Question AS q, Opinión AS o WHERE q.number BETWEEN 01 AND 19 GROUP BY q.issue HAVING mostOf o.professorId = $sel2`

```
AND o.period = $sel3 ARE NOT EXIST
(SELECT r FROM Response AS r WHERE
r.serial <> o.serial  OR r.number
<> 25 OR r.value is not high) WITH
CALIBRATION $sel4;
```

Additionally, the formal specification of requirement *C3* is normalized using the implication equivalence as:

C3 = { c.* | c$\in$ Course $\wedge$ c.departmentId = \$sel1 : (mostOf o: o $\in$Opinion $\wedge$ o.period = \$sel3 : $\neg$($\exists$r: r $\in$ Response : r.serial$\neq$o.serial $\vee$ r.number $\notin$ {22,23,24} $\vee$ (r.number = 22 $\wedge$ $\neg$(r.value is high)) $\vee$ (r.number = 23 $\wedge$ $\neg$(r.value is low)) $\vee$ (r.number = 24 $\wedge$ $\neg$(r.value is high))))}.

Trans(*C3*) is
```
SELECT c.* FROM Course AS c,
Opinión AS o WHERE c.departmentId =
$sel1 GROUP BY c.code HAVING mostOf
o.period = $sel3 ARE NOT EXIST
(SELECT r FROM Response AS r WHERE
r.serial <> o.serial OR NOT r.number
IN {22,23,24} OR (r.number = 22 AND
r.value is not high) OR (r.number = 23
AND r.value is not low) OR (r.number
= 24 AND r.value is not high)) WITH
CALIBRATION $sel4;
```

5. CONCLUSIONS

Currently, many methodologies exist for software development. These methodologies adequately regard several issues of database applications, such as user interfaces, communication with other systems, data insertion, and so on. However, existing methodologies have not been thought to build database applications that involve fuzzy terms. Thus, we have focused on incorporating these kinds of requirements.

Information requirements for decision-making systems may include terms whose nature is vague. These terms represent user's preferences. The fuzzy set theory allows for the logical-mathematical representation of fuzzy terms. This allows for one to handle them computationally. To satisfy those information requirements with vague linguistic terms, the standard database query language SQL has been extended by using the fuzzy logic developed in previous works. In this sense, SQLf has emerged as one of these extensions.

On the other hand, the fuzzy query language SQLf has been used in the development of several applications. Based on these experiences, in this paper we have proposed a method to develop database applications that support fuzzy requirements. We presented a methodological way for incorporating fuzzy queries in software development in order to support user requirements involving linguistic terms expressing preferences. Firstly, the analysis produces a list of fuzzy requirements in a natural language. Secondly, each fuzzy requirement is formally specified by means of a fuzzy logic extension of tuple calculus. Thirdly, formal specified fuzzy requirements are translated into SQLf for their implementation in a real DBMS. The proposed translation is this paper's main contribution. We have restricted the use of fuzzy quantifiers in tuple fuzzy calculus to those expressions that may be translated into SQLf.

To the best of our knowledge, none of the previous existing database application development methods consider fuzzy queries formal specification. That is our contribution. The use of formal specification techniques in our method avoids the ambiguity of natural language. Even when user requirements involve vague nature linguistic terms, thanks to the use of fuzzy logic, ambiguity is avoided.

We have developed a case study. The application consists of querying the Student Opinion Survey of Simon Bolivar University solving user requirements involving linguistic terms. In this way, we have shown the boundaries of our proposed method.

Since our method involves formal specifications of fuzzy queries and includes translation rules from the formal specification of fuzzy queries to SQLf language, it is feasible to develop a computer-aided tool in order to formally specify fuzzy queries and generate an SQLf code. Thus, development of applications with fuzzy requirements may be automated and the ambiguity problem may be eliminated or minimized. As a future work, we plan to automate the translation from natural language into tuple calculus. This work opens a way for a new generation of information systems with support on SQLf fuzzy querying language.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the help of our friend and colleague Graciela Perera for this work revision. Also we give thanks for the wisdom given to us by our best friend Jesus Christ, the Lord. "If any of you lacks wisdom, you should ask God, who gives generously to all without finding fault, and it will be given to you." (Jas. 1:5, New International Version).

REFERENCES

[1] Zadeh, L. A., Fuzzy Sets, Information and Control, 8(3), pp.338-353, 1965.

[2] Bosc, P., and Pivert, O., SQLf: A Relational Database Language for Fuzzy Querying, IEEE Transactions on FuzzySystems, 3, pp.1-17, 1995.

[3] Galindo, J., Urrutia, A., and Piattini, M., Fuzzy Database Modeling, Design and Implementation, Idea Group Publishing, 2006.

[4] Bordogna, G., and Psaila, G., Customizable Flexible Querying Classic Relational Databases. En: Handbook of Research on Fuzzy Information Processing in Databases (Eds. J. Galindo J), Hershey, PA, USA: Information Science, pp.189-215, 2008.

[5] González, C., Goncalves, M., and Tineo, L., A New Upgrade to SQLf: Towards a Standard in Fuzzy Databases. 20th Proceeding of International Workshop on Database and Expert Systems Application. pp.442-446. 2009.

[6] Muñetón, A., Zapata, C. M., and Arango, F., Reglas para la Generación Automática de Código definidas sobre Metamodelos Simplificados de los Diagramas de Clases, Secuencias y Máquina de Estados de Uml 2.0, DYNA, Año 74, pp.153, 267-283, 2007.

[7] Goncalves, M., Rodríguez, R., and Tineo L., Incorporando Consultas Difusas en el Desarrollo de Software, Revista Avances en Sistemas e Informática, 6 (2), pp.87-101, 2009.

[8] Rodríguez, R., and Goncalves M., Implementación de requisitos difusos en sistemas orientados a datos utilizando el lenguaje OCL y lógica difusa, Enl@ce: Revista Venezolana de Información, Tecnología y Conocimiento, 8 (1), pp.31-54, 2011.

[9] Codd, E., Relational Completeness of Data Base Sublanguages; Data Base Systems, Courant Computer Science Symposia Series; Prentice-Hall: Englewood Cliffs, NJ, 6, 1972.

[10] Lacroix, M., Pirotte, A. Domain-Oriented Relational Languages, Proceeding of VLDB, pp.370-378, 1977.

[11] Galindo, J., Medina, J.M. and Aranda, M.C., Querying Fuzzy Relational Databases Through Fuzzy Domain Calculus, International Journal of Intelligent Systems, 14 (4), pp.375-411, 1999.

[12] Buckles, B.P., Petry, F.E., and Sachar, H.S., A Domain Calculus for Fuzzy Relational Databases, Fuzzy Sets and Systems, 29, pp.327-340, 1989.

[13] Takahashi, Y., A Fuzzy Query Language for Relational Databases. Fuzziness in Database Management Systems (Eds. P. Bosc, J. Kacprzyk), Physica Publisher, Heidelberg, pp.365-384, 1995.

[14] Liétard, L., Contribution à l'Interrogation Flexible de Bases de Données: Étude des Propositions Quantifiées Floues [Thèse de Doctoract]. Université de Rennes I, France,1995.

[15] Mokhtari, A., Pivert, O., Hadjali, A., and Bosc, P., Towards a route planner capable of dealing with complex bipolar preferences. Proceeding of the 12th IEEE Conf. on Intelligent Transportation Systems, USA, pp.556–561, 2009.

[16] Zadeh, L. A., PRUF – A Language for the Representation of Meaning in Natural Languages. Proceeding of the 5th IJCAI, Cambridge, MA, pp.395-460, 1977.

[17] Tineo, L., Interrogaciones Flexibles en Bases de Datos Relacionales, Trabajo de Ascenso, Universidad Simón Bolívar, Caracas, Venezuela, 1998.

[18] Goncalves, M., and Tineo, L., SQLfi y Sus Aplicaciones, Revista Avances en Sistemas e Informática, 5 (2), pp.33-40, 2008.

[19] Gries, D., and Schneider, F. B., A Logical Approach to Discrete Math. Texts and Monographs in Computer Science. Springer-Verlag, 1994.

[20] Tineo, L., A fuzzy quantifiers' interpretation for database querying, Proceedings of FIP, 423-428, 2003.

[21] Kawash, J., Complex quantification in Structured Query Language (SQL): a tutorial using relational calculus, Journal of Computer in Mathematics and Science Teaching, 2004.