# UFleSe: User-Friendly Parametric Framework for Expressive Flexible Searches
# UFleSe: Cadre paramétrique conviviale pour des recherches expressives et flexibles

Mohammad Halim Deedar and Susana Muñoz-Hernández

***Abstract*— We present a parametric framework (UFleSe) with a user-friendly interface having a search engine that enables regular users (without the need of neither technical nor theoretical knowledge) to define their fuzzy concepts, rules, similarity relations, synonyms, antonyms, and personalizing their definitions for different users, and to link them with the crisp database fields for performing flexible, expressive queries in a language close to natural language. It works over multiple modern and conventional data formats, such as JSON, SQL, Prolog, CSV, XLS, and XLSX. We present the syntax involved in the construction of our various flexible searching criteria and their personalizations. Furthermore, we present the architecture of this novel system that combines fuzzy, crisp data, and similarity relations in its queries to return constructive answers ordered by a degree of searching criteria satisfaction (truth-value between 0 and 1). Finally, we include a comparative analysis of different fuzzy querying systems here, and we provide various experiments, to show the system behavior, performance, efficiency, and scalability as well.**

***Résumé*— Nous présentons un cadre paramétrique (UFleSe) avec une interface conviviale ayant un moteur de recherche qui permet aux utilisateurs réguliers (sans connaissances ni techniques ni théoriques) de définir leurs concepts flous, règles, relations de similitude, synonymes, antonymes et personnaliser de leurs définitions pour différents utilisateurs et de les relier aux champs d'une base précise de données pour effectuer des requêtes expressives flexibles dans une langue proche du langage courant. Il fonctionne avec plusieurs formats de données modernes et conventionnels, tels que JSON, SQL, Prolog, CSV, XLS et XLSX. Nous présentons la syntaxe impliquée dans la construction de nos différents critères de recherche flexibles ainsi que leurs personnalisations. De plus, nous présentons l'architecture de ce nouveau système qui combine des données floues et précises ainsi que des relations de similitude dans ses requêtes pour renvoyer des réponses constructives ordonnées par échelle de satisfaction des critères de recherche (valeur de vérité entre 0 et 1). Enfin, nous incluons ici une analyse comparative de différents systèmes de requête floue et nous proposons diverses expériences pour montrer également le comportement, les performances, l'efficacité et la notion évolutive du système.**

***Index Terms*— Framework, fuzzy criteria, fuzzy logic, personalization of criteria, search engine, similarity relation.**

## I. Introduction

WHAT is a flexible query, and why do we need it? Assume, a database storing information about different employees, such as their name, age, major, and so on, by flexible query we mean queries, such as "I am looking for a good employee" or "I am looking for a manager with quite a lot of experience and not very old." In many cases, users do not have enough details about the exact characteristics of the elements they want to search in a database. In addition, the users are, in general, unable to give the proper values of the data characteristic corresponding with their searching preferences. In those situations, the systems that allow performing flexible queries can be beneficial to satisfy their needs. Moreover, searching in a fuzzy way not only retrieves the exact information we are looking for from a database (having crisp information), but, it also provides all the possible and available information very close to the criteria we have set for our query. To clarify, we take an example of a database of *restaurants* having crisp information. If we execute a query in a crisp way for retrieving restaurants with a menu price lower than 10 euros, and there is one restaurant with a menu price of 10.15 euros, then this one will not be retrieved, while it might be the one we are looking for. Moreover, the personalization of the fuzzy criteria is one of the most important features when searching. There are situations, such as having a database and multiple users who can use a searching system and access the database, to perform flexible queries. Suppose we have a "cheap" fuzzy predicate defined in our system for retrieving restaurants with low-cost menu prices from the database. Thus, personalization can provide results based on the levels of satisfaction of each user for the same criteria. Fuzzy logic [1], [2] has substantiated its capability devoted to the management of vague information in a different number of applications (such as control systems,

Manuscript received September 4, 2019; revised December 13, 2019; accepted January 8, 2020. Date of current version August 21, 2020. *(Corresponding author: Mohammad Halim Deedar.)*

The authors are with the Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, 28660 Madrid, Spain (e-mail: halim.deedar@alumnos.upm.es; susana@fi.upm.es).

Associate Editor managing this article's review: Mohamed Hamed Abdelpakey.

Digital Object Identifier 10.1109/CJECE.2020.2966733

TABLE I
EMPLOYEES DATABASE

| Name | Age | Major |
|---|---|---|
| Davis | 70 | law |
| Thomas | 61 | Commerce |
| Scott | 45 | Economics |
| Martin | 20 | Computer Science |

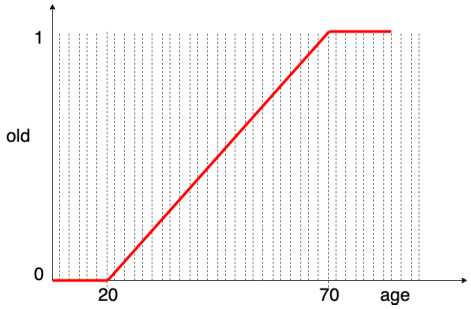



Fig. 1. *Old* fuzzification function.

database, or expert systems). The integration of fuzzy logic and logic programming [3] provides fuzzy logic programming with the capability of dealing with ambiguity and approximate reasoning. This integration provided us the facility to program the applications in order to understand the fuzzy characteristics (it is cold), fuzzy rules (if it is cold, turn on the heater) and fuzzy actions (since it is not too cold, turn on the heater at medium degree). Therefore, in our approach, we use Prolog, which is more declarative and a successful programming language for representing knowledge, and the Fuzzy logic that defines not only if an individual belongs to a set or not but it also provides the degree of its belonging to that set. Supposing, a database of "employees" in Table I and the definition of the function "old" in Fig. 1 with respect to the value of "age," and the question "is employee X is old?" with Fuzzy logic we can deduce that "Davis" is "very" old, "Thomas" is "quite" old, "Scott" is "hardly" old, and "Martin" is "not" old at all. We highlight the words "very," "quite," "hardly," and "not" because the usual answers for the query are "1," "0.82," "0.5," and "0" for the individuals Davis, Thomas, Scott, and Martin, respectively, and the humanization of the crisp values is done in the same way by defuzzification.

In this article, we describe our parametric framework that allows users (without knowledge of the low-level syntax of the framework and knowledge about Prolog) to define their fuzzy searching criteria (fuzzy concepts, fuzzy rules with or without credibility values, and conditional or unconditional default values), similarity relations, personalization of the fuzzy criteria, synonyms and antonyms for performing expressive, and flexible searching over multiple modern and conventional database tables. The framework takes care of all the mapping processes to link between the crisp information stored in the database and the fuzzy concept without the user interventions. The syntax involved in the construction of our flexible queries and the details about the design and implementation of our system for processing flexible queries is described in this article. As proof of concept, we also report a thoughtful example of the behavior of our system by experimenting with flexible searches over two databases. This article is structured as follows. A brief overview of state of the art and the details about the background works on which we depend on is explained in Section I. Afterward, we present syntactical constructions and the architecture of the system in Section II. Implementation details about the framework user interface are given in Section III. Three examples of system behavior, performance, efficiency, and scalability are included in Section IV. In Section V, we add a comparative analysis of the main features of this approach and the main fuzzy querying systems in the literature. Finally, several conclusions and future works are discussed in Section VI.

### A. State of the Art of Fuzzy Query Systems

We have reviewed the systems that allow flexible queries over databases in the literature to establish the basis of the proposal presented in this contribution.

*1) Fuzzy Query Systems:* Obtaining fuzzy answers by posing fuzzy queries over databases having nonfuzzy information has been studied in some works, where good revisions can be found in the works, such as the ones provided by Bosc and Pivert [4], Dubois and Prade [5], Tahani [6], and Rodriguez [7], although maybe a little bit outdated. Moreover, the integration of fuzzy logic and logic programming resulted in the development of many fuzzy systems over Prolog. To represent fuzzy knowledge, we need to create a link between fuzzy and nonfuzzy concepts, and to achieve that, we could use any of the existing frameworks. Apart from the theoretical frameworks as [8], we know about the Prolog-Elf system [9], the F-Prolog language [10], the FRIL Prolog system [11], the FLOPER [12], the FuzzyDL reasoner [13], the Fuzzy Prolog system [14], [15], or RFuzzy [16]. All of them somehow implement the fuzzy set theory introduced by Zadeh [1], and they allow us to execute the connectors required to retrieve the nonfuzzy information stored in databases. Moreover, there are some tools for accessing regular databases having nonfuzzy information in a fuzzy way, such as by Bosc and Pivert [4], [18], Ribeiro and Moreira [17], Bordogna and Pasi [19], [21], and Kacprzyk and Zadrozny [20]. Furthermore, many promising proposals for performing fuzzy queries on relational databases can be found in [22]–[27] with the syntax somewhat similar to SQL, or in some work, they have extended the SQL language. In Section V, we provide a qualitative comparison analysis, among different fuzzy querying systems devoted to the main features existing in different approaches.

### B. Background

In this section, we describe the earlier works on which our framework is based on.

*1) RFuzzy Library:* RFuzzy[1] [16] was implemented as a library of Ciao Prolog [28] to increase the expressiveness of Prolog with the possibility of managing fuzzy information.

[1]In RFuzzy's name, the "R" means real, because the truth-value that it uses is a real number instead of an interval or union of intervals as in Fuzzy Prolog. RFuzzy should not be confused with the term "R-Fuzzy set," which means rough fuzzy set.

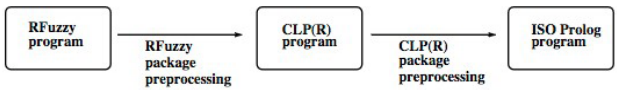



Fig. 2. RFuzzy architecture.

The goal that led to developing RFuzzy was mainly to reduce the complex syntax of Fuzzy Prolog [14], for representing fuzzy reasoning with truth-values represented as real numbers. The answers provided by the RFuzzy are never constraints (as in Fuzzy Prolog); it can be differentiated between the crisp and fuzzy predicates without the intervention of the user, and it can manage the introduction of truth-values. What makes RFuzzy distinctive is that it allows defining types and default values (general and conditional). It is the first tool that provided credibility to the rules through the implementation of multi-adjoint logic. RFuzzy structure is shown in Fig. 2. In general, RFuzzy contains the following features that make it different from other approaches.

1) It represents the truth-values using real numbers instead of unions of intervals between the real number (that is the complex representation of Fuzzy Prolog).
2) It provides a concrete syntax to define types so that the user does not need to code them in Prolog.
3) The results provided by RFuzzy are not constraints; instead, it provides direct constructive results.
4) It allows the implementation of both the default and partial default values.
5) RFuzzy uses multiadjoint logic to model its semantic.

*a) Semantics:* The multiadjoint algebra presented in [29]–[34] is used to give semantics to RFuzzy. The purpose of using these semantics is that it provides credibility for the rules. Comprehensive details about the semantics can be found in the cited articles. With the help of this semantics, we are able to use the maximum operator to decide between multiple rules results and to get the less generic rule as the valid one instead of the one with the higher value. Refer to the article [35] for details related fuzzy logic program conditioned by the combination of truth and a priority value.

*b) Rule Syntax:* The general structure that is used to define RFuzzy rules according to a multiadjoint logic semantics is shown in 1 [14]

$$P(\text{args}P_j,\ V_j) \xleftarrow{(Pr_j, V_{cj})\&_i} @_j\ (Q_1\ (\text{args}Q_{1j}, V_{1j})) \ldots (Q_n\ (\text{args}Q_{nj}, V_{nj}\ )) \quad (1)$$

where $P$ is the predicate, and $j$ is one of the definitions from a set of definitions $j \in [1,N]$ (where N is the number of rules that we have to define for predicate $P$, and $j$ identifies one of these rules). argsP$_j$ are the arguments of $P$ in the rule $j$, in the same way, argsQ$_i$ are the arguments of $Q_i$, where $i \in [1,n]$ and $n$ is the number of elements of the body of the clause. $V_{\text{ij}}$ is the truth-value of $Q_i$(argsQ$_{\text{ij}}$,$V_{\text{ij}}$). $@_j$is the aggregation operator of the rule $j$. $V_{\text{cj}}$ is the credibility to calculate the truth-value, Pr$_j$ is the priority of $j$ rule with respect to other rules of $P$ definition, and $\&_i$ is the aggregation operator (*product*), for example, for obtaining the truth value with the credibility of the rule.

*2) FleSe Framework:* FleSe [36] is the former version of UFleSe, which is a handy framework for performing fuzzy and nonfuzzy queries over Prolog databases containing crisp information. It uses the RFuzzy package, which is a Prolog library developed with Ciao Prolog, and it uses Fuzzy logic with Prolog. FleSe was not user-friendly enough to be used by regular users (users without knowledge of Prolog). It was limited on querying over Prolog data files only, and users must understand all the syntax behind the flexible queries, and they should define the fuzzy criteria manually inside the Prolog file outside the system before performing searches (it was almost writing a program in the syntax we explain in Section II). We overcome all these limitations with UFleSe by introducing new features that we explain in Sections II and III.

## II. Description of System

UFleSe is a search tool that provides a user-friendly Web interface for users to be able to make expressive queries (using fuzzy searching criteria, fuzzy rules, synonyms, antonyms, similarity, negation, and fuzzy qualifiers) over conventional and crisp data. UFleSe allows users to upload their data to define in an easy way the similarity concepts and the fuzzy criteria (fuzzy concept, rules, synonyms, and antonyms) that they want to use for searching. UFleSe lets the different users personalize these fuzzy search criteria according to their personal preferences, and it provides constructive answers to the queries.

### A. Framework Syntax

We present here the syntactical constructions behind various operations of our framework. They are defined according to RFuzzy [16] syntax.

*1) Database Definition Syntax:* In order to define a link between the fuzzy predicate and the database, first of all, we have to know what is stored inside each database field.

The syntax, which is responsible for outlining the contents of a database into concepts, that we use in our searches is shown in 2. In the syntax, $P$ is the name of the database table, $A$ is its arity, $N_i$ is the name assigned to a column (field) of the database table where values are of type $T_i$ (boolean_type, enum_type, integer_type, float_type, string_type), and $i \in$ [1,A] identifies each field of the table. We give an example in 3, where we have defined an employee database having six columns

$$\text{define\_database}(P/A, [(N_i, T_i)]) \quad (2)$$

```
define_database(employee/6,
(name, string_type),
(age, integer_type),
(years_of_experience, integer_type),
(years_of_studying, integer_type),
(major, enum_type),
(distance_to_the_workcenter, integer_type)]).
```
(3)

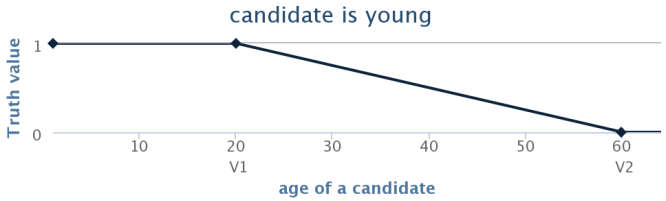



Fig. 3. Decreasing function format for young fuzzification criterion.

*2) Fuzzification Syntax:* The process of linking the non-fuzzy values of our table with a fuzzy predicate (criterion) is called fuzzification. The syntax presented in 4 is responsible for defining fuzzification. It computes the fuzzy value for a fuzzy predicate from the crisp value existed in any column of the database. The breakpoints define the fuzzification function, and the rest of the values are easily deduced to provide the truth-value of the fuzzy criterion of each domain value of the crisp field. In the syntax, $P$ and $N_i$ mean the same as in 2, fPredName is the name for the fuzzy predicate that is defined, and *[valIn, valOut]* is a list of pairs of values to define the break points domain of the fuzzification. To clarify the syntax, we give an example in 5, in which we define the youth of an employee based on his/her *age*

$$\text{fPredName}(P) : \sim \text{function}(P, (N_i), [(\text{valIn}, \text{valOut})]). \tag{4}$$

$$\text{young(employee)} : \sim \text{function(age(employee)}, [(20, 1), (60, 0)]). \tag{5}$$

By defining this fuzzy criterion (young), we mean the age till which the employee is very young is 20 with truth-value 1, and as employee's age is increasing, they get older, and the truth-value decreases, and the minimum age on which an employee is not at all young is 60 with the truth-value 0.

In FleSe, the breakpoints for the domain of fuzzification were not fixed; therefore, the formats of the potential fuzzy criteria in the Prolog file could be wrongly defined by users without any knowledge about the functions. In order to make the fuzzification process easy for regular users, we have considered three different formats (increasing, decreasing, and medium) for modeling the structure of the potential fuzzy criteria. In this way, user can easily define the fuzzy criteria using the breakpoints of the domain of fuzzification ([valIn, valOut]), where *valIn* includes the criteria breakpoints ($V1$, $V2$, $V3$, and $V4$) that gets its value as input by the user and *valOut* holds the truth-value (0 or 1) which is going to be assigned automatically by the framework based on the type of the fuzzy criteria.

We provide an example in Fig. 3, in which the criterion young has a decreasing format that has been defined by the user using two breakpoints ($V1 = 20$) and ($V2 = 60$), as it has decreasing format; therefore, the system assigns a truth-value 1 for the breakpoint $V1$ and a truth-value 0 for the breakpoint $V2$. Hence, while searching for young candidates, the system computes the candidates with the age smaller and equal to 20 with a degree of satisfaction 1, which means the candidates are completely young, and the candidates with age greater and equal to 60 with a degree of satisfaction 0, which means the candidates are not young at all. The candidates with age between the domain $V1$ and $V2$ will have a different degree of satisfaction (truth-value between 0 and 1), which keeps decreasing as the age of the candidate gets increased. Just like that, our system takes care of the *increasing* and *medium* format functions, as shown in Figs. 4 and 5, respectively. We explain later in the upcoming sections, how to define these fuzzy searching criteria.

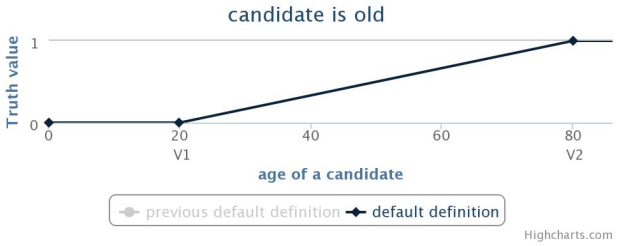



Fig. 4. Increasing function format for old fuzzification criterion.

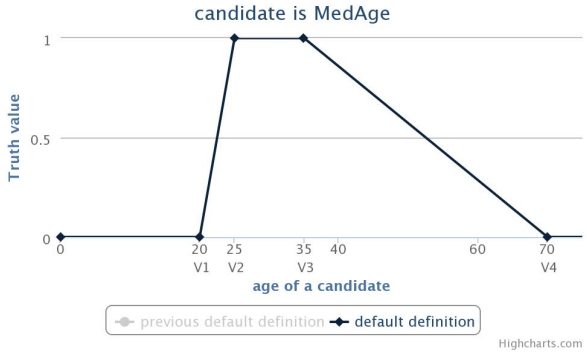



Fig. 5. Medium function format for medAge fuzzification criterion.

*3) Defining Fuzzy Rules Syntax:* Rules help us to define more expressive, flexible searching criteria by defining the satisfaction of a fuzzy criterion (predicate) from the satisfaction of other fuzzy criteria. We can define a fuzzy rule with a single body (having one fuzzy predicate in its body), as shown in 7, or with multiple bodies, as shown in 8. In 7, *aggr* is the aggregator[2] used to combine the truth-values of the fuzzy predicates in fuzzyPred, which is just a conjunction of names of fuzzy predicates. In both of them, the fuzzyPred is name of the fuzzy predicate, $P$ means the same as in 2 and fPredName the same as in 4. We show an example in 8 in which we say, "an employee is a good employee if he/she is young, highly experienced, and with high qualification" at the same time

$$\text{fPredName}(P) : \sim \text{rule(fuzzyPred)} \tag{6}$$

$$\text{fPredName}(P) : \sim \text{rule(aggr,fuzzyPred)} \tag{7}$$

$$\begin{aligned}\text{good\_employee(employee)} : \sim\ & \text{rule(prod, (rather(young(employee)),} \\ & \text{highly\_experienced(employee),} \\ & \text{very(high\_qualification(employee))))).}\end{aligned} \tag{8}$$

[2]Available aggregation operators are *min* for minimum, *max* for maximum, *prod* for the product, luka for the Łukasiewicz operator, *dprod* for the inverse product, and *dluka* for the inverse Łukasiewicz operator and complement.

*4) Syntax for Handling Missing Information:* If there is missing information or the content of the database has a null value, we can face unexpected behavior which does not satisfy the definition of our fuzzy predicate from a value stored in a database. To avoid such problem, the syntax (shown in 9) is used for defining the satisfaction of the fuzzy concepts which we call it "defining default truth-values of the fuzzy concepts," and, its primary objective is not to stop a derivation process even if a value is missing. In the syntax, $P$ means the same as in 2, fPredName the same as in 4, and TV is a truth-value (a float number between 0 and 1). We give two examples in 10 and 11 which indicates, in the absence of information, we consider that the employee is not experienced (this is what the zero value means) and that, in the absence of information, the employee is considered to be medium old

$$\text{fPredName}(P) : \sim \text{default\_to}(TV) \tag{9}$$

$$\text{experienced(employee)} : \sim \text{default\_to}(0) \tag{10}$$

$$\text{old(employee)} : \sim \text{default\_to}(0.5). \tag{11}$$

*5) Synonyms and Antonyms Definition Syntax:* Synonyms and antonyms help us to increase the number of fuzzy criteria that can be used for querying the database by defining a fuzzy predicate from another fuzzy predicate. With the syntactical constructions in 12 and 13, we can define a fuzzy predicate as a synonym or antonym of other fuzzy predicates. For example, we can define "old" from "aged" as a synonym and "young" from "old" as an antonym, and so on. The examples are shown in 14 and 15. In the syntax, fPredName is the name of the fuzzy predicate (young, old, aged, and so on), $P$ is the same as in 2, credOp is the operator (*product* by default), and credVal is the credibility (a type float number, which is 1 by default)

$$\text{fPredName}(P) : \sim \text{synonym\_of}(\text{fPredName2}(P), \text{credOp}, \text{credVal}) \tag{12}$$

$$\text{fPredName}(P) : \sim \text{antonym\_of}(\text{fPredName2}(P), \text{credOp}, \text{credVal}) \tag{13}$$

$$\text{aged(employee)} : \sim \text{synonym\_of}(\text{old(employee)}, \text{prod}, 1) \tag{14}$$

$$\text{young(employee)} : \sim \text{antonym\_of}(\text{old(employee)}, \text{prod}, 1). \tag{15}$$

*6) Syntax for Defining Credibility to The Rules:* We can define credibility for the fuzzy rules and concept by adding the syntax in 16 as a tail in their syntactical constructions, together with the required operator to combine it with its truth-value. In the syntax, credVal is the credibility (a number of float type) and credOp is the credibility operator.[3] We show an example in 17 in which we say that the employee *Sara Guzman* is young with a truth-value of "0.6," but this rule has the credibility of "0.8," and the operator that must be used to combine the credibility with the truth-value is the *minimum*

$$\text{with\_credibility}(\text{credOp}, \text{credVal}) \tag{16}$$

$$\text{young(employee)} :\sim \text{value}(0.6)\text{if (name(employee)is equal to sara guzman) with\_credibility}(\text{min}, 0.8). \tag{17}$$

*7) Similarity Relations Syntax:* Sometimes, the users might be interested in searching for some items in the database table that have a characteristic similar to a particular characteristic. The syntax in 18 is responsible for defining similarity between values in RFuzzy. In the syntax, $P$ and $N$ mean the same as in 2, TV is the same as in 9, and $V1$ and $V2$ are two values for the column $N$ of the database table $P$. In the example, in 19, we say that "the employees with major Business is 0.7 similar to the commerce one"

$$\text{similarity\_between}(P, N, [(V1), N(V2), \text{TV}]) \tag{18}$$

$$\text{similarity\_between(employee, major(Commerce), major(Business), 0.7).} \tag{19}$$

*8) Syntax for Personalization of Fuzzy Concepts:* The tail syntax in 20 is added for the personalization of a fuzzy rule. During searches, the personalized definition of the searching criteria will be used for users that have defined their specific definition. The general definition will be used otherwise. In the syntax, *Username* is the name of any user (a string). We provide an example in 21 in which we say that "Sara considers that the employee Luis is not old." Therefore, if it is she who poses a query to the system asking for old employees, she will not obtain *Luis*

$$\text{only for user 'UserName'} \tag{20}$$

$$\text{old(employee)} :\sim \text{value}(0) \text{ if (name(employee) is equal to luis) only for user 'Sara'.} \tag{21}$$

### B. System Architecture

Our system is a Web application implemented in Java with a Prolog search engine. Java and Ajax are used for the development of the Web user interface with lots of improvements in communication with Prolog. Our database is managed directly by the Prolog code because it has the facilities for linking to it. We are not restricted to any database or database interface. It uses CLP(R) and RFuzzy packages, which are Prolog libraries developed for Ciao Prolog.

The type of database that our system works with is a table that can be in any of the conventional and modern formats (XLS, XLSX, JSON, CSV, SQL, and Prolog).

The system architecture, which is shown in Fig. 6 has four main parts.

*1) System Modules:* UFleSe contains six different modules.

1) *Parsing Module:* It is responsible for parsing the data-files (database table) of different formats (XLS, XLSX, JSON, CSV, SQL, and Prolog) and generate a configuration file (Prolog format).

[3]The credibility operators are the conjuntor, such as product, Łukasiewicz conjunctor and Gödel conjunctor which are mathematical functions which is monotone and nondecreasing in their coordinates. Users can easily use them in our framework by selecting "prod," "luka," and "min" for the field "credOp" in the interface.

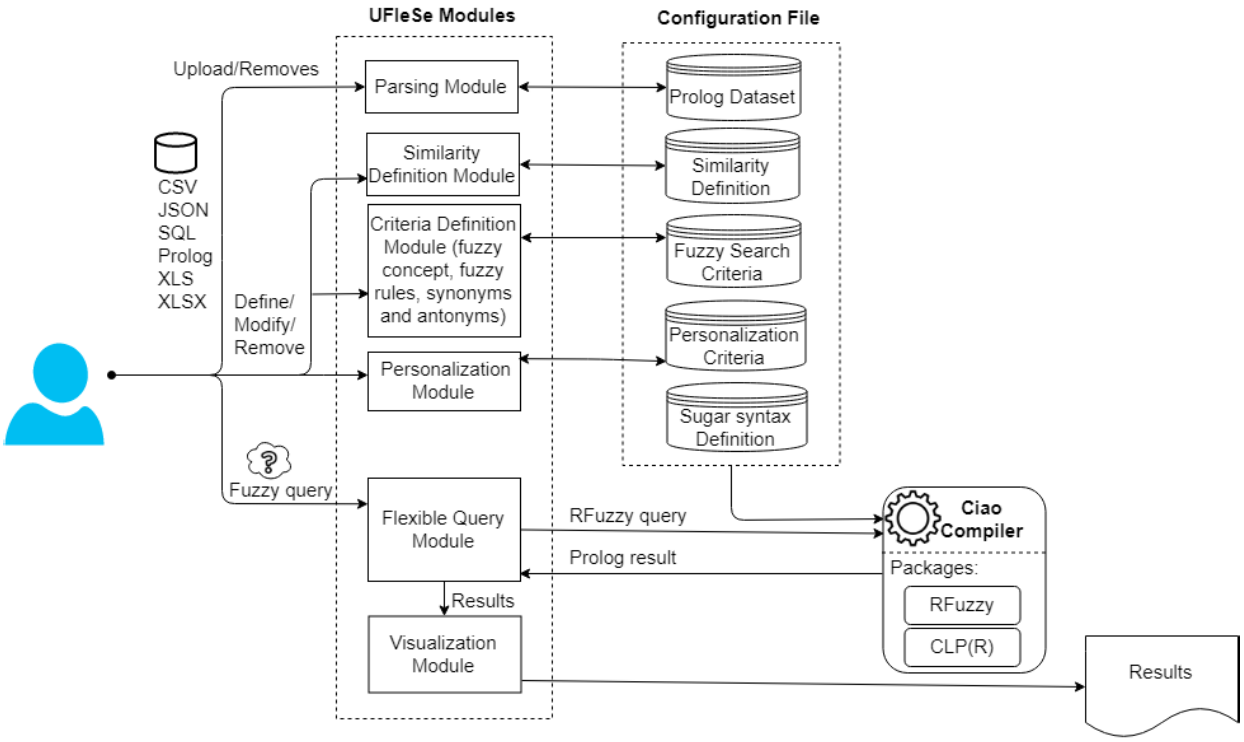



Fig. 6. UFleSe architecture.

2) *Criteria Definition Module:* This module is responsible for the fuzzification of the criteria. It allows users to define the fuzzy criteria (fuzzy concepts and fuzzy rules definitions) by creating a link between the fuzzy predicate and the crisp information stored in the data file, or by defining one fuzzy predicate from another fuzzy predicate (synonyms and antonyms).
3) *Similarity Definition Module:* This module is responsible for defining the similarity relation between different elements of the database table.
4) *Personalization Module:* This module is responsible for the personalization of the fuzzy criteria.
5) *Flexible Query Process Module:* It takes the flexible queries as input (which is a natural language like query) from the users and provides the results as output (constructive with truth-value) after several computations of fuzzy criteria and similarity relations. We discuss more the system behavior while posing a flexible query in *Experimentation* (Section IV).
6) *Visualization Module:* It is responsible for the visualization of the resulting data sets in order to make the results more useful and understandable to the users.

*2) Configuration File:* The configuration file contains the Prolog data set, which holds the contents of our database table and all the flexible search criteria (fuzzy concepts, fuzzy rules, similarity relations, personalization of fuzzy predicates, and synonyms and antonyms of fuzzy predicates) defined by the users through the framework interfaces. The syntactic sugar is added directly to the configuration file by the system during the creation of the file, and it contains the negation operator and modifiers. That is, it contains all the information needed for resolving the queries.

### C. Ciao Compiler

The compiler of the Ciao System [28] is the engine that provides a result for the queries by using the translation of the expressive, flexible query to RFuzzy syntax. It uses the RFuzzy and CLP(R) libraries (packages) and consults data of the configuration file.

### D. Result

Our framework provides constructive results with different degrees of accomplishment (truth-values) assigned.

## III. Framework User Interface

In order to remove the gap between the advanced users (researchers and developers) and the regular users (without having knowledge of Prolog and low-level syntax of the framework), we provide a user-friendly interface that feeds from the knowledge stored inside the framework. In this way, regular users can easily experiment and perform flexible, expressive searches devoted to their defined fuzzy searching criteria over their database tables, which they have in any of conventional and modern formats, we present more comprehensive details about the conversion process of the uploaded data files in [37]. Before anything else, the user must sign in to the system using his/her Gmail or Twitter account and upload the database tables.[4] Once uploaded, the user has to define the types of the columns (integer, string, float, enum, and date) existing in the database table. Afterward, the system generates the configuration file. We present the uploading interface in Fig. 7,

[4]The type of database that our system works with is a table, that can be in any of the conventional and modern formats (CSV, XLSX, XLS, JSON, SQL, and Prolog).

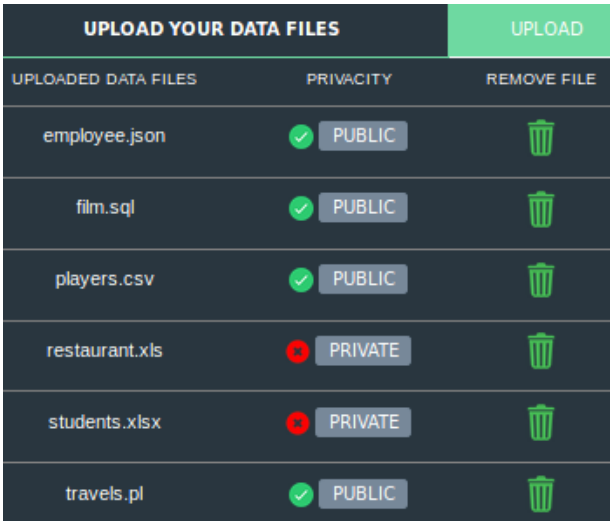



Fig. 7. Data-file uploading interface.

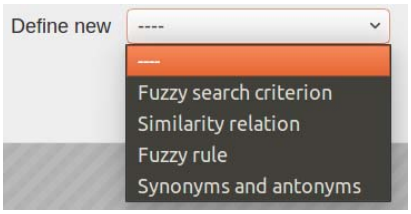



Fig. 8. Defining new criteria.


Fig. 9. Defining fuzzy search criteria.

in which we can see a list of database tables in various formats, and the option "Privacity" that can be either "private" or "public." Each user decides the accessibility to the other users of the database tables that he/she has uploaded.

To make the fuzzification process easy for regular users, we provide them different interfaces to use for defining various flexible search criteria and other features by selecting one of the options from the combo shown in Fig. 8.

### A. Defining Fuzzy Search Criteria (Fuzzy Concept)

The interface for defining fuzzy criteria is shown in Fig. 9. For more details about this interface, refer to [38]. For ease of use, we have fed most of the interface variables with the information stored inside the configuration file so that users can define fuzzy searching criteria easily. This interface defines fuzzy searching criteria based on the syntax in 4.

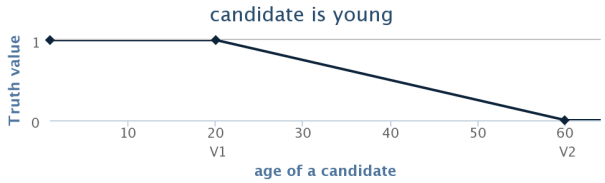



Fig. 10. Young fuzzy predicate graph.

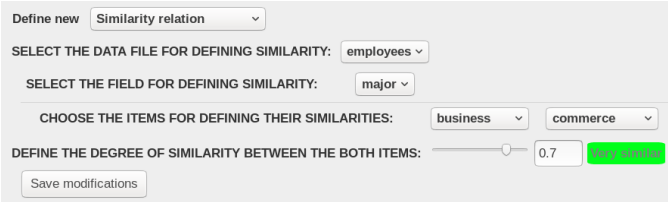



Fig. 11. Defining similarity relations.

*1) Step 1:* In the interface, user has to select the database *table name*, and the *column* of crisp data, such as employee table, and the column *age*.

*2) Step 2:* Once selected, then the user should define the *name* of the fuzzy predicate in the text field area, for example, young.

*3) Step 3:* Afterward, the user has to define the *formats* of the criteria. There are three common formats for searching fuzzy criteria based on a crisp field of data. 1) Increasing criterion (the value of the criterion increases when the value of the crisp increases). 2) Decreasing criterion (the value of the fuzzy criterion decreases when the value of crisp data increases). 3) Range criterion (the value of the fuzzy criterion has truth-value 1 in a range of crisp data, increases before and decrease after that range).

Based on the example, the user selects the second option for the young fuzzy predicate, which is, "LESS age MORE young." Once selected, the system will define the structure of the fuzzy criteria by asking a few simple questions in a language easy to understand by regular users. The user's answers define the domain of fuzzification by fixing the breakpoints of the fuzzy function ($V1$, $V2$, $V3$, and $V4$). Based on our example, the system asks for *the maximum age in which the employee is completely young*, we give 20, and *the minimum age in which employee is not at all young*, we give 60, the system takes care of the definition of the truth-values to complete the domain of the fuzzification (by assigning truth-value 1 till the age 20 and truth-value 0 from the age 60). By saving the form, the fuzzy predicate "young" gets created inside the configuration file, and we can start posing a flexible query based on the criterion "young." We present the graphical representation of "young" fuzzy predict in Fig. 10 which is a decreasing format graph.

### B. Defining Similarity Relation Interface

The interface for defining similarity relation devoted to the RFuzzy syntax in 18 is shown in Fig. 11.

*1) Step 1:* The first thing the user needs to do is, to select the *table name* and the (*column*) of the values between which he/she wants to define the similarity relation. In the example (see Fig. 11), the user has selected the table employee and the column *major*.

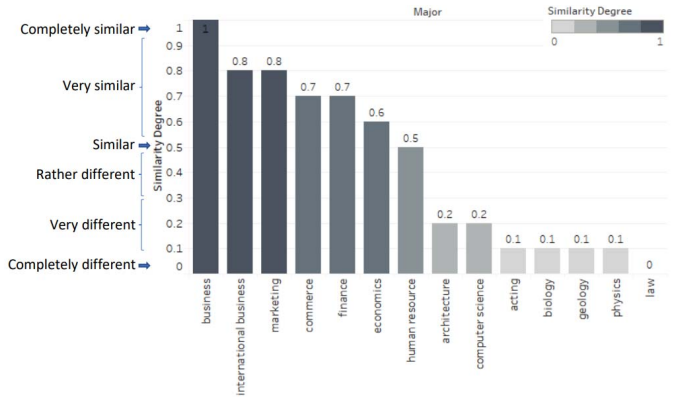



Fig. 12. Graph of similarity relations between *business* and other majors.

*2) Step 2:* Afterward, the user chooses the items from the lists (of values of major field) between which he wants to define the similarity relation. (in the example user has selected the values *business* and *commerce* from the column *major*.)

*3) Step 3:* Finally, once items are selected, the user must define a similarity degree (a truth-value between 0 and 1) that represents how similar these two items are.

By saving the form, the similarity relation between *business* and *commerce* gets created in our system, and we can start posing flexible queries based on "majors similar to business" or "majors similar to commerce." We provide more details about the similarity interface in [39].

Our system does not allow contradictions while the definition of a similarity relation which already exists in the system. Suppose if there is a similarity relation in the system as "business is very similar to marketing" (with a truth-value 0.8 for example) and a user wants to define a new relationship between the same objects, but with a different similarity degree, such as "business is very different from marketing" (with a truth-value 0.3 for example), in this case, instead of defining a new relation our system will update the prespecified relation with the new similarity degree that the user has recently defined. Moreover, our system will take care of the symmetric relation between the similarity relations. Taking an example, if a user has already defined a similarity relation, such as "business is very similar to marketing," then the user does not need to define another symmetric relation as "marketing is very similar to business" when he/she wants to retrieve queries based on "marketing".

While defining a similarity relation degree, we have considered six different categories (completely different, very different, rather different, similar, very similar, and completely similar) based on different similarity degrees to make the user understand what we exactly mean by these real numbers between 0 and 1. To clarify, we present a graph in Fig. 12 to show similarity relations and their categories between different values of the field major of employee database[5] with respect to the value "business."

[5]The defined relations are just examples, we are not saying that business is 0.8 similar to marketing or any other relations. You need to add another clause with that information if you want to say that too.

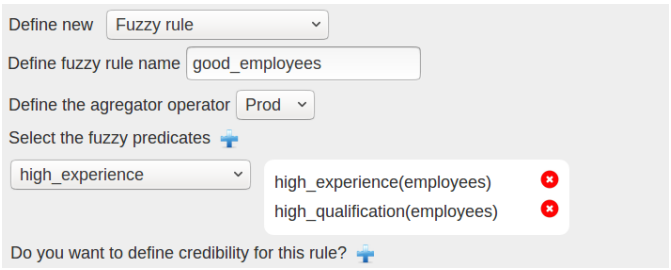



Fig. 13. Defining fuzzy rule criteria.

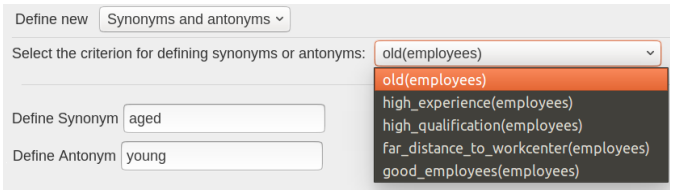



Fig. 14. Defining synonyms and antonyms.

### C. Defining Fuzzy Rules Criteria Interface

In order to do more expressive fuzzy searches, our system allows users to define fuzzy searching rules through which users can define a fuzzy rule based on the satisfaction of other fuzzy predicates. To clarify, we give an example. Suppose we have already defined a few fuzzy search criteria for querying about *highly_qualified* and *highly_experienced* employees. Now, there might be some employees who are *highly_qualified* (having the highest truth-value) but not *highly_experienced* in comparison with other employees who are *highly_qualified* (with 0.5 truth-value out of 1), but more *experienced*. Therefore, a novel system allows a single-flexible query that considers all these two criteria (or maybe more) at the same time while searching for a *good employee* and provides the most satisfactory and efficient result. We provide the interface in Fig. 13 for defining a fuzzy rule, where, users need to define a *name* for the fuzzy rule, the *aggregation operator* (prod, min, and max luka), afterward, he has to define the *fuzzy predicates* from the combo list which are going to be used as body of the fuzzy rule. Moreover, users can define the *credibility* for the fuzzy rule by clicking the plus symbol for the credibility option.

### D. Defining Synonyms and Antonyms Interface

We can define synonyms and antonyms using the interface shown in Fig. 14 in which from the combo, we select the fuzzy predicate on which we want to define synonyms or antonyms of it. We provide an example user interface (see in Fig. 14) where we have defined *aged* as a synonym and young as an antonym of the *old* fuzzy predicate of the table employee. Synonyms and antonyms of a fuzzy criterion can be defined at the same time or separately based on the user's choice.

### E. Personalization Interface

In order to let the users have more satisfactions on their searching results devoted to a fuzzy search criterion, we provide them the personalization option that allows the user to

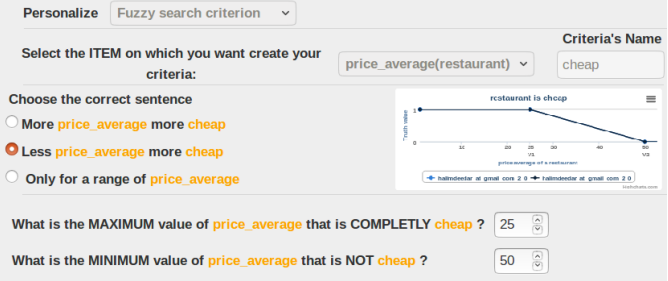



Fig. 15. Personalization of fuzzy criteria.

modify the definition of a fuzzy concept to be better adapted to his/her preferences. With this feature, users can personalize how the framework translates the nonfuzzy attributes stored in the database into the fuzzy ones that he/she uses in queries. We present the personalization interface in Fig. 15 in which it asks the user which fuzzy predicate he/she wants to personalize and his/her preferences for the fuzzification of the values stored in the database. The interface for personalization is user-friendly enough; the user only needs to define his/her preferred values for the breakpoints of the domain of the fuzzification. Our system uses the personalized predicate if the user logged in has previously defined a personalization; otherwise, it uses the general definition. We provide a working example of this functionality with details using our system in Section IV-B.

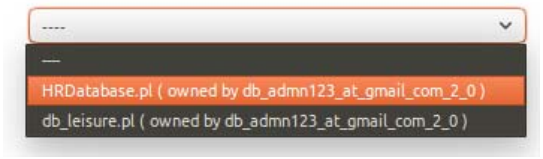
### *F. Search Engine*

Our framework gets the user's query as input according to the following format:

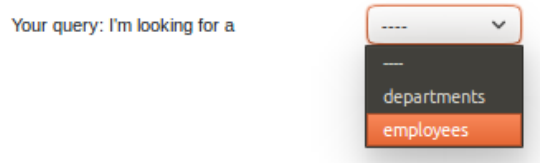
$$\text{I am looking for a/an } \boxed{\text{individual}} \left\{ \begin{array}{l} \boxed{\text{not}}\ \boxed{\text{m}}\ \boxed{\text{fp}} \\ \text{whose } \boxed{\text{nfp}}\ \boxed{\text{co}}\ \boxed{\text{value}} \end{array} \right\} \boxed{\text{AND}} \tag{22}$$

where *individual* is the name of the table that we are querying (e.g., employee), *not* gives a negation mechanism to the queries, *m* is the modifier (fairly, little, rather, very, and very little), *fp* is a fuzzy predicate from the fuzzy searching criteria defined in the system for a data table, such as fast, cheap, big, expensive, . . ., *nfp* is a nonfuzzy predicate corresponding a crisp field of the table, such as name, club, age, price, . . ., *co* indicates comparison operand, and it consists of an operand (is equal to, is different from, is bigger than, is lower than, is bigger than or equal to, is lower than or equal to, and is similar to), and *value* is a crisp value of the field *nfp*. To pose a query over a database, first, we select the *configuration file* that is the already uploaded data file enriched with the searching criteria definition [see Fig. 16(a)] and then the *database table* [see. Fig. 16(b)] existed inside the configuration file. Each configuration file can contain various tables. Each query is compound by so many simple queries as wanted (they can be added by the AND functionality).

One of the most interesting features of the user interface is that it interacts with the system, and therefore, it knows if the attribute selected in the search engine [see Fig. 17(a)] is fuzzy or not. We provide an example in [Fig. 17(b)] for

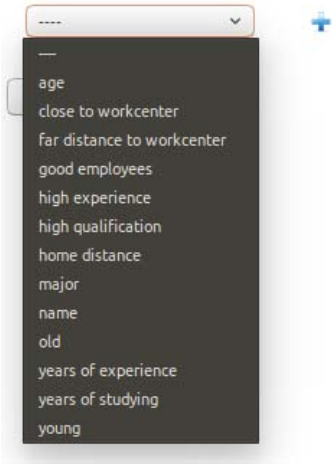



(a)

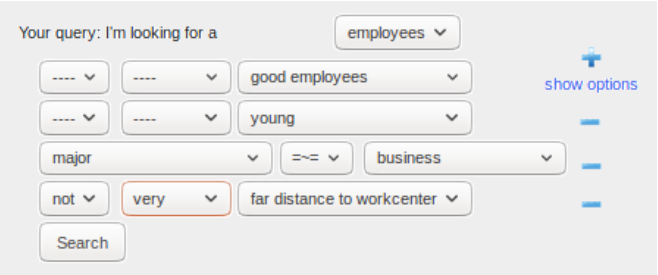



(b)

Fig. 16. Selecting the configuration file and the database table. (a) Selecting the configuration file. (b) Select what we are looking for.




(a)




(b)

Fig. 17. Selecting the attributes and posing the flexible query. (a) Selecting the predicates. (b) Search engine.

posing a flexible (compound) query *"I am looking for a good employee, young with a major similar to business and with home not very far from the work center."* In the search engine, the first and second selected criteria (good employees and young) are the fuzzy predicates; therefore, two boxes in the left appears for defining *not* (negation) operator and the *modifiers* are selected (if needed), and the third criterion is a nonfuzzy predicate (major); therefore, two boxes appear in the right for defining the *comparison operand*. The other box is a combo for selecting a *crisp value* from the database table, or if the operand is not a similarity operator, then a free text field appears for entering a crisp value. The fourth criterion is

| 1 | name | age | years of experience | years of studying | major | distance to the workcenter | driving license |
|---|---|---|---|---|---|---|---|
| 2 | Fran perez | 23 | 1 | 6 | geology | 25 | 0 |
| 3 | Sara Guzman | 32 | 8 | 6 | business | 15 | 1 |
| 4 | Maria Luisa Gonzalez | 48 | 20 | 4 | commerce | 18 | 0 |
| 5 | Manuela Fernandez | 34 | 8 | 10 | physics | 5 | 1 |
| 6 | Martin Herranz | 20 | 0 | 2 | computer science | 30 | 1 |
| 7 | Luis Rodriguez | 25 | 2 | 6 | economics | 9 | 0 |
| 8 | Javier Casas | 30 | 5 | 5 | acting | 21 | 0 |
| 9 | Marta Suarez | 21 | 1 | 6 | biology | 52 | 1 |
| 10 | Smith | 46 | 20 | 2 | human resource | 43 | 1 |
| 11 | Johnson | 33 | 8 | 6 | Marketing | 15 | 1 |
| 12 | Williams | 38 | 10 | 4 | commerce | 18 | 0 |
| 13 | Jones | 64 | 32 | 10 | international business | 5 | 1 |
| 14 | Davis | 70 | 50 | 8 | law | 25 | 1 |
| 15 | Wilson | 20 | 1 | 5 | finance | 21 | 0 |
| 16 | Moore | 31 | 8 | 6 | economics | 52 | 1 |
| 17 | Taylor | 65 | 40 | 2 | architecture | 43 | 1 |
| 18 | Anderson | 66 | 33 | 6 | business | 15 | 1 |
| 19 | Thomas | 61 | 40 | 4 | commerce | 18 | 0 |
| 20 | Allen | 24 | 2 | 8 | law | 25 | 1 |
| 21 | Scott | 36 | 10 | 6 | economics | 9 | 0 |
| 22 | Nelson | null | 11 | 6 | architecture | 52 | 1 |

Fig. 18. *Employees* database example.

again a fuzzy predicate, where we have selected *not* negation operator and the modifier *very*. The plus sign serves to add more conditions to the query; it is the AND functionality (it only has a line at the beginning), the minus sign removes the added condition from the query and the *show options* label through which we switch the operator for combining the truth-values from minimum to product, Łukasiewicz or any other. The attributes are the names we give to the columns using the syntax in 2 and the fuzzy predicates are defined using the syntaxes in 4, 12, and 13 (the system does not allow to define duplicate fuzzy predicate, even we defined it manually inside the configuration file; it appears only once). After posing the query, we need to press the button *search*. The search engine then provides the query results, grouped in five tabs: "10 best results," "results over 70," "results over 50," "results over 0," and "all results." This helps the user to choose the results that best fit his/her query.

### G. Available Release

We have implemented an open-source prototype of our framework. We have created a project in a forge to make our software available online[6] for users and researchers with a free software license GNU.

## IV. Experiments

In this section, we present the system behavior, performance, efficiency, speed, and scalability through three experiments using two databases *employees* database (having 21 tuples) and *players* database (having 1000 tuples).

### A. Experimentation 1

We have analyzed the system behavior when a flexible query is executed through this example. We have developed a small database about *employees* for experimentation purposes only. We have shown a brief schema of this database in Fig. 18. We posed an expressive, flexible query in the search engine, which is in a natural language like query [see Fig. 17(b)]: "I am looking for good employees who are young with a major similar to business and whose distance to work center is not very far." This flexible query is a compound query which contains fuzzy concepts, fuzzy rules, antonyms, similarity relations (based on the 4, 7, 13, and 18), negation, and *very* modifiers in its structure, and it implies the definition of the five attribute domains in the database table, such as years_of_studying, years_of_experience, age, major, and home_distance attributes. To fuzzify them, we have defined the following fuzzy criteria on them, such as *highly_qualified, highly_experienced, young, similarity relations between majors, and far_distance_to_workcenter*, respectively. The database searching process starts when the fuzzy query is executed in the search engine. The flexible query process module translates the fuzzy query into an RFuzzy query with the help of the Ciao compiler. Afterward, it divides the query into different subqueries, calculates and combines all the results obtained from each subquery at the same time, and provides us the final result.

1) *Subquery 1 (Fuzzy Rule):* "I am looking for good employees." This query is a fuzzy rule based on the syntax in 7, which has two more fuzzy predicates in its structure which are *highly_qualified* and *highly_experienced*. The result obtained from this subquery is shown in Table II.
2) *Subquery 2 (Antonyms):* " I am looking for young employees." This query is solved by calculation of the antonym of fuzzy predicate "old," which are defined using 4 and 13. The result obtained from this subquery is shown in Table III.
3) *Subquery 3 (Similarity Relations):* " I am looking for majors similar to business." This query provides all the employees whose majors are similar to business (including, of course, the employees with a business major). This query is based on 18. The result obtained from this subquery is shown in Table IV.

[6]UFleSe's download link: https://github.com/FuzzyLP/UFleSe.git

TABLE II

RESULT OF:"*Good Employees*"

| name | years of experience | years of studying | truth value |
|---|---|---|---|
| davis | 20 | 8 | 0.8 |
| anderson | 20 | 6 | 0.6 |
| jones | 11 | 10 | 0.55 |
| thomas | 18 | 4 | 0.36 |
| nelson | 11 | 6 | 0.33 |
| scott | 10 | 6 | 0.3 |
| sara guzman | 8 | 6 | 0.24 |
| johson | 8 | 6 | 0.24 |
| moore | 8 | 6 | 0.24 |
| maria luisa | 10 | 4 | 0.2 |
| williams | 10 | 4 | 0.2 |
| taylor | 18 | 2 | 0.18 |
| manuela | 4 | 6 | 0.12 |
| javier casas | 5 | 5 | 0.12 |
| allen | 2 | 8 | 0.12 |
| smith | 10 | 2 | 0.1 |
| luis rodriguez | 2 | 6 | 0.06 |
| fran perez | 1 | 6 | 0.03 |
| marta suarez | 1 | 6 | 0.03 |
| wilson | 1 | 5 | 0.03 |
| martin herranz | 0 | 2 | 0 |

TABLE III

RESULT OF:"*Young Employees*"

| name | age | truth value |
|---|---|---|
| martin herranz | 20 | 1 |
| wilson | 20 | 1 |
| marta suarez | 21 | 0.99 |
| fran perez | 23 | 0.96 |
| allen | 24 | 0.94 |
| luis rodriguez | 25 | 0.93 |
| javier casas | 30 | 0.86 |
| moore | 31 | 0.84 |
| sara guzman | 32 | 0.83 |
| johson | 33 | 0.81 |
| manuela | 34 | 0.8 |
| scott | 36 | 0.77 |
| williams | 38 | 0.74 |
| smith | 46 | 0.63 |
| maria luisa | 48 | 0.6 |
| thomas | 61 | 0.41 |
| jones | 64 | 0.37 |
| taylor | 65 | 0.36 |
| anderson | 66 | 0.34 |
| davis | 70 | 0.29 |
| nelson | null | 0 |

TABLE IV

"*Majors Similar to Business*"

| name | major | truth value |
|---|---|---|
| sara guzman | business | 1 |
| anderson | business | 1 |
| johson | marketing | 0.8 |
| jones | int_business | 0.8 |
| maria luisa | commerce | 0.7 |
| williams | commerce | 0.7 |
| wilson | finance | 0.7 |
| thomas | commerce | 0.7 |
| manuela | economics | 0.6 |
| luis rodriguez | economics | 0.6 |
| moore | economics | 0.6 |
| scott | economics | 0.6 |
| smith | humResource | 0.5 |
| martin herranz | ComputerSc | 0.2 |
| taylor | architecture | 0.2 |
| nelson | architecture | 0.2 |
| fran perez | geology | 0.1 |
| javier casas | acting | 0.1 |
| marta suarez | biology | 0.1 |
| davis | law | 0 |
| allen | law | 0 |

TABLE V

*"Employees Not Far From Work Center"*

| name | home distance | truth value |
|---|---|---|
| manuela | 10 | 0.89 |
| allen | 12 | 0.87 |
| sara guzman | 15 | 0.83 |
| marta suarez | 15 | 0.83 |
| maria luisa | 18 | 0.8 |
| wilson | 18 | 0.8 |
| martin herranz | 20 | 0.78 |
| javier casas | 21 | 0.77 |
| thomas | 21 | 0.77 |
| taylor | 24 | 0.73 |
| fran perez | 25 | 0.72 |
| johnson | 26 | 0.71 |
| moore | 26 | 0.71 |
| scott | 27 | 0.7 |
| williams | 28 | 0.69 |
| jones | 32 | 0.64 |
| anderson | 32 | 0.64 |
| nelson | 32 | 0.64 |
| luis rodriguez | 35 | 0.61 |
| smith | 43 | 0.52 |
| davis | 45 | 0.5 |

4) *Subquery 4 (Negation Operator)*[7]*:* "I am looking for employees who are not far distance from work center." We have used the "not" negation modifier to experiment with its working. This query provides all the employees who have less distance to the work center. This query is based on the negation of 4. The result obtained from this subquery is shown in Table V.

Our flexible query is calculated using the following functions.
old(employee): ∼ function(age(employee), [(20, 0)], [(90, 1)]).
young(employees):∼ antonym_of(old(employees), prod, 1).
highly_experienced(employees):
∼function(years_of_experience(employees)
, [(0, 0),(15,1)]).
highly_qualified(employees): ∼function(years_of_studying
(employees), [(0,0),(10,1)]).
far_distance_to_workcenter(employees): ∼ function(home_distance
(employees), [(0,0),(90,1)]).
good_employees(employees):∼ rule(prod,(highly_experienced
(employees), highly_qualified(employees)))with_credibility
(prod,1).
define_similarity_between(employee,
major(business),major(business),1).
define_similarity_between(employee, major(business), major
(marketing), 0.8). define_similarity_between(employee,
major(business), major
(international_business),0.8). define_similarity_between
(employee,major(business), major(commerce), 0.7).
define_similarity_between
(employee,major(business), major(finance), 0.7).
define_similarity_between
(employee,major(business), major(economics), 0.6).
define_similarity_between
(employee,major(business), major(human_resource), 0.5).

[7]The "not" negation operator of our system allows users to negate any fuzzy search criteria without writing any additional syntax or code inside the configuration file.

10 best results | Results over 0% | All results

| employees | name | age | years of experience | years of studying | major | home distance | Truth Value |
|---|---|---|---|---|---|---|---|
| nº.1 | jones | 64 | 11 | 10 | international business | 32 | 0.37 |
| nº.2 | thomas | 61 | 18 | 4 | commerce | 21 | 0.36 |
| nº.3 | anderson | 66 | 20 | 6 | business | 32 | 0.34 |
| nº.4 | scott | 36 | 10 | 6 | economics | 27 | 0.3 |
| nº.5 | sara guzman | 32 | 8 | 6 | business | 15 | 0.24 |
| nº.6 | johnson | 33 | 8 | 6 | marketing | 26 | 0.24 |
| nº.7 | moore | 31 | 8 | 6 | economics | 26 | 0.24 |
| nº.8 | maria luisa | 48 | 10 | 4 | commerce | 18 | 0.2 |
| nº.9 | williams | 38 | 10 | 4 | commerce | 28 | 0.2 |
| nº.10 | taylor | 65 | 18 | 2 | architecture | 24 | 0.18 |
| nº.11 | manuela | 34 | 4 | 6 | economics | 10 | 0.12 |
| nº.12 | javier casas | 30 | 5 | 5 | acting | 21 | 0.1 |
| nº.13 | smith | 46 | 10 | 2 | human resource | 43 | 0.1 |
| nº.14 | luis rodriguez | 25 | 2 | 6 | economics | 35 | 0.06 |
| nº.15 | fran perez | 23 | 1 | 6 | geology | 25 | 0.03 |
| nº.16 | marta suarez | 21 | 1 | 6 | biology | 15 | 0.03 |
| nº.17 | wilson | 20 | 1 | 5 | finance | 18 | 0.03 |
| nº.18 | martin herran: | 20 | 0 | 2 | computer science | 20 | -0 |
| nº.19 | davis | 70 | 20 | 8 | law | 45 | -0 |
| nº.20 | allen | 24 | 3 | 8 | law | 12 | -0 |
| nº.21 | nelson | null | 11 | 6 | architecture | 32 | -0 |

Fig. 19. Result of query: "*I am looking for good employees who are young with a major similar to business and whose distance to work center is not far.*".

define_similarity_between (employee,major(business), major(computer_science), 0.2). define_similarity_between (employee,major(business), major(architecture),0.2). define_similarity_between (employee,major(business), major(geology), 0.1). define_similarity_between (employee,major(business), major(physics), 0.1). define_similarity_between (employee,major(business, major(acting,0.1). define_similarity_between (employee,major(business), major(biology), 0.1). define_similarity_between(employee,major(business), major(law), 0).

The flexible query process module with the help of Ciao compiler combines all the four subqueries using the conjunction AND (conjunctions are defined based on the type of the queries posed in the search engine), considering the priorities between different rules while calculation processes, and returns a final result, as shown in Fig. 19.

We can observe that: finally (see Fig. 19), "Jones" with the age of 64, 11 years of experience, 10 years of studying, and with the major "international business," and home distance "32 km," is the best candidate with the highest truth-value "0.37" that satisfies the requirements of our query. It is a very restricted search based on compound criteria, and therefore, the truth-value is quite low.

### *B. Experimentation 2*

In this experimentation, we are going to test the system behavior for the personalization of fuzzy searching criteria.

| employees | name | age | Truth Value |
|---|---|---|---|
| nº.1 | martin herranz | 20 | 1 |
| nº.2 | wilson | 20 | 1 |
| nº.3 | marta suarez | 21 | 0.99 |
| nº.4 | fran perez | 23 | 0.96 |
| nº.5 | allen | 24 | 0.94 |
| nº.6 | luis rodriguez | 25 | 0.93 |
| nº.7 | moore | 31 | 0.84 |
| nº.8 | sara guzman | 32 | 0.83 |
| nº.9 | johnson | 33 | 0.81 |
| nº.10 | manuela | 34 | 0.8 |
| nº.11 | scott | 36 | 0.77 |
| nº.12 | williams | 38 | 0.74 |
| nº.13 | smith | 46 | 0.63 |
| nº.14 | maria luisa | 48 | 0.6 |
| nº.15 | thomas | 61 | 0.41 |
| nº.16 | jones | 64 | 0.37 |
| nº.17 | taylor | 65 | 0.36 |
| nº.18 | anderson | 66 | 0.34 |
| nº.19 | davis | 70 | 0.29 |
| nº.20 | javier casas | 30 | -0 |
| nº.21 | nelson | null | -0 |

Fig. 20. Result of young predicate after Personalization by User1.

*User1:* personalizes the young fuzzy predicate, which we have defined previously in Section IV-A over *employees* database, saying that "the maximum age on which employees are completely young is 20, and the minimum age on which employees are not at all young is 90." But, *user1* wants to personalize this fuzzy predicate, considering that *Javier Casas with age 30 is not young* (supposed, any physical problem have been taken into account). After querying over "young" fuzzy criterion by *User1*, in contrast with the result set of the query through "admin user" (any other user without personalization of the young definition), as shown in Table III, where *Javier Casas* was retrieved with truth-value "0.86," which means he is very young. Now, we have queried the same database using the same "young" fuzzy predicate by the user "User1," and we retrieved "Javier Casas" assigned with truth-value "0," which means he is not young at all (for user1 opinion). The result obtained from the personalized query is shown in Fig. 20. Hence, any number of users can personalize this fuzzy criterion, without having any contradiction while querying for young employee. Moreover, the system does not duplicate any fuzzy predicate after the personalization; therefore, while posing a query, there is always a distinct fuzzy and nonfuzzy predicate in the list [see in Fig. 17(a)]. The system is "intelligent" enough (user sensitive) to determine which user has personalized the predicate so that it provides results based on his/her personalized criteria if he/she is the one who poses the query.

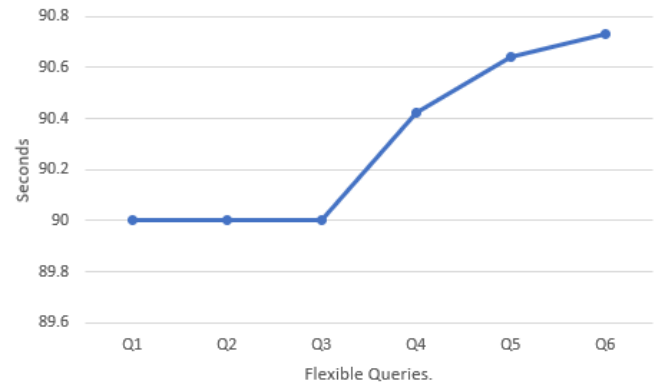
PlayersDatabase.json*


```
[
  {
    "ID": 158023,
    "Name": "L_ Messi",
    "Age": 31,
    "Skill Moves": 4,
    "Position": "RF",
    "Acceleration": 91,
    "Agility": 91,
    "height": 144,
    "Nationality": "Argentina",
    "Overall": 94,
    "Potential": 94,
    "Club": "FC Barcelona",
    "Weak Foot": 4,
    "Real Face": "Yes",
    "Jersey Number": 10,
    "Contract Valid Until": "2021",
    "Weight": "159lbs",
    "Stamina": 72,
    "Strength": 59
  },
```

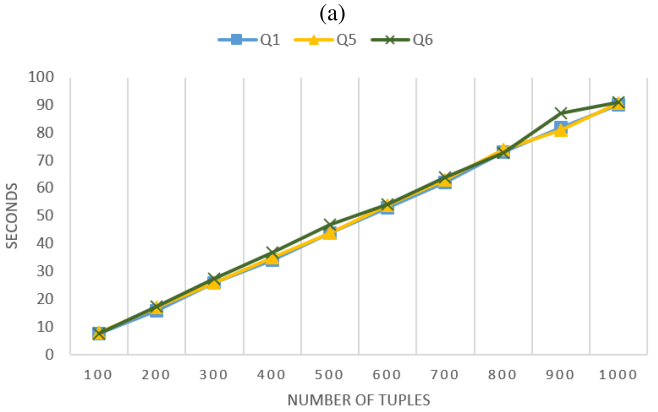

Fig. 21. Example of players database.

TABLE VI
SET OF FLEXIBLE QUERIES

| ID | Flexible query | Rows | Time calculated |
|---|---|---|---|
| Q1 | Looking for high stamina players | 1000 | 90 |
| Q2 | Looking for weak players | 1000 | 90 |
| Q3 | Looking for players whose positions are similar to ST | 1000 | 90 |
| Q4 | Looking for top players whose | 1000 | 90.42 |
| Q5 | Looking for good players | 1000 | 90.64 |
| Q6 | Looking for top players | 1000 | 90.73 |




(a)




(b)

Fig. 22. Execution times in queries performed on the *players* DB. (a) Execution time. (b) Scalability in execution time.

TABLE VII
EXECUTION TIMES VARYING NUMBER OF TUPLES

| N.tuples | Q1 | Q5 | Q6 |
|---|---|---|---|
| 100 | 7.61 | 7.95 | 7.78 |
| 200 | 16 | 17 | 17.45 |
| 300 | 26 | 26 | 27.53 |
| 400 | 34 | 35 | 37 |
| 500 | 44 | 44 | 47 |
| 600 | 53 | 54 | 54.21 |
| 700 | 62 | 63 | 64 |
| 800 | 73 | 74 | 72.80 |
| 900 | 82 | 81 | 87 |
| 1000 | 90 | 90.64 | 91 |

### C. Experimentation 3

We performed different tests to measure the system efficiency on a database about FIFA football players, which consists of 1000 tuples. For the sake of the brevity, we have shown a partial view of the database (see Fig. 21). In order to analyze the efficiency of our system, we have performed two different tests on the database: 1) the first one is by varying the complexity of the query. 2) Second is by varying the number of tuples computed on the same query, to analyze the system scalability.

We designated a set of different queries to measure the performance of the system. We presented these queries, that vary in complexity, in Table VI along with the number of tuples calculated and their execution time in seconds. In Fig. 22(a), we can see that the complexity of the query does not have a big effect on the execution time (only a little bit).

We measured scalability by varying the number of tuples in the database from 100 to 1000. To do that, we have executed the most representative queries. Execution times are shown in Table VII, and we have shown them in Fig. 22(b). In Fig. 22, we can notice how the scalability increases devoted to the number of computed rows in the most complex queries, that the ones with more computational needs.

## V. COMPARATIVE ANALYSIS

In Table VIII, we present a comparison between the characteristics of the main fuzzy querying systems in the literature and our proposal.

The first thing that makes our proposal differs from other fuzzy query systems is the use of the RFuzzy library in our framework, which makes our syntax for fuzzifications very simple. Moreover, it allows us to provide default truth-values (conditioned and unconditioned), credibility to the rules, and the priorities features that help us in increasing the capabilities in deciding which result is more preferred to be chosen among the result provided by the different rules, it is not necessary if the last rule offers a result with higher truth-value. Moreover, the proposals by Bosc and Pivert [4], [18],

TABLE VIII
COMPARISONS BETWEEN FEATURES OF FUZZY QUERY SYSTEMS

| Model | Ribeiro[17] | Bosc[4] | Bordogna[21] | Kapcprzyk.[20] | Martinez.[26] | UFleSe |
|---|---|---|---|---|---|---|
| Modifiers | X | X | | | X | X |
| Negation operator | | | | | | X |
| Personalization of fuzzy predicates | | | | | | X |
| Query over multiple data formats | | | | | | X |
| Natural language like query | X | | | | X | X |
| Similarity relations | | | | | X | X |
| Graphical definition of fuzzy search criteria. | | | | | | X |
| Priorities for the rules | | | | | | X |
| Credibility for the rules | | | | | | X |
| Definition of Synonyms | | | | | | X |
| Definition of antonyms | | X | | | | X |
| Parametric framework | | | | | | X |
| Search engine for expressive queries | | | | | | X |
| Simple query syntax | | | | | | X |

Ribeiro and Moreira [17], Bordogna and Pasi [19], [21], and Kacprzyk and Zadrozny [20], most of them are without any implementation associated, it is only a theoretical description, and some of them not seem to be a search engine project. Furthermore, several proposals allow querying relational databases in a fuzzy way; some of the promising proposals can be found in [22]–[27]. Most of the work discussed in these articles focus in advancing the effectiveness of the existing methods, in adding new syntactic constructions in the query or in allowing to introduce the conversion between nonfuzzy values and fuzzy values required to perform the query, for which they use a syntax very similar to SQL or an extension of SQL (they are perfectly adequate for advanced users, developers, or researchers), but they are a bit difficult to the regular users (the syntax is a little bit complicated for them), and in order to get the answers for the queries, the user has to instruct the search engine how to get the fuzzy results from nonfuzzy values stored in the database, and the user must also understand the low-level syntax and the semantics of the language, and he/she must know about the structure of the database tables. As far as we know, the features that make our framework novel and different from others are as follows.

1) Personalization of fuzzy predicates for flexible searches for different users.
2) A user-friendly system that allows any users without knowing the low-level syntax of the framework or without knowing Prolog can define fuzzy searching criteria for performing flexible searches.
3) A search engine with a general form for getting the user's query (not only a text field area) and understand them, and retrieves the possible results even though the condition given in the query does not satisfy that, therefore, for the restaurant's example before, the food with the price "10.15" euro would be in the valid results set.
4) It allows performing flexible searches based on synonyms and antonyms.
5) It provides credibility to the rules that make our search engine that first engine that implements multiadjoint logic.
6) It provides constructive results with the truth-value so that users can understand the result data set.
7) It provides the "not" negation modifier, which can negate any fuzzy concept and rules without writing any extra code.
8) Allowing users to perform crisp, fuzzy, and compound queries all together in a single query interface.

In contrast with most of the fuzzy query systems, there are several approaches in fuzzy logic devoted to the similarity representation, but with many difference with our proposal, as the works in [40]–[43], their most significant differences with ours are: 1) that we do not force our relations to be an equivalence for our similarity criteria. Since some of them mention, this is too limiting for real-world applications. Moreover, 2) we are not trying to evaluate the closeness (or similarity) between two fuzzy propositions. We handle between predicates only two kinds of similarity: 1 (for synonyms) and 0 (for antonyms), but no other values for similarity can be defined between fuzzy concepts. They could be defined, but we do not consider it friendly for users. We let the user define similarities between the different values of a field of the database so that we can search for answers with similar values to a concrete value of a field.

## VI. CONCLUSION

In this article, we describe the easy procedure that lets us define fuzzy concepts and link them to crisp database fields. We have presented UFleSe, a parametric framework that allows the user to perform fuzzy and nonfuzzy queries over conventional and modern databases by linking the nonfuzzy values which are stored in the database with fuzzy concepts. The system translates expressive queries into enrich Prolog queries (with RFuzzy syntax) and provide answers for them. It handles if there are null values existed inside our database as well.

We have explained the syntax and semantics behind various operations of our framework, along with the structure of the system and its implementation details.

Our main goal is to provide a system user-friendly enough so that regular users (without knowledge of the low-level syntax of the framework and about programming) can define their fuzzy concepts, rules, similarity relations, synonyms, and antonyms. It allows users to personalize the fuzzy concepts for performing expressive, flexible searches (fuzzy, crisp, and compound queries) over their databases and let other users be

able to search their data in the same expressive way that they have defined. The rest of the users can also personalize the fuzzy search criteria definitions if they want. We have provided three case studies to prove the concept and to show the behavior, performance, efficiency, and scalability of our system. We hope this contribution helps in the development and the improvement of more human-oriented search mechanisms so that we use human-oriented attributes (young, expensive, and so on) instead of computer-oriented ones (age under X, price over Y, and so on).

Our article is oriented to validate our framework with different users interested in searching over their data by uploading their databases, adding the fuzzy criteria and similarity definitions for expressive searching in their data, searching and using them in our framework, and finally, providing their opinion related their satisfaction with it.

We are also working on introducing a mechanism for clustering the system users devoted to their profile and searching records.

## REFERENCES


[1] L. A. Zadeh, "Fuzzy sets," *Inf. Control*, vol. 8, no. 3, pp. 338–353, 1965.

[2] L. A. Zadeh, "Calculus of fuzzy restrictions," in *Fuzzy Sets and Their Applications to Cognitive and Decision Processes*. New York, NY, USA: World Scientific, 1974, pp. 1–39.

[3] J. W. Lloyd, *Foundations of Logic Programming*, 2nd ed. Berlin, Germany: Springer-Verlag, 1987.

[4] P. Bosc and O. Pivert, "SQLf: A relational database language for fuzzy querying," *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 1, pp. 1–17, Feb. 1995.

[5] D. Dubois and H. Prade, "Using fuzzy sets in flexible querying: Why and how?" in *Flexible Query Answering Systems*, A. Troels, C. Henning, and L. H. Legind, Eds. Boston, MA, USA: Springer, 1997, pp. 45–60. [Online]. Available: http://dl.acm.org/citation.cfm

[6] V. Tahani, "A conceptual framework for fuzzy query processing—A step toward very intelligent database systems," *Inf. Process. Manage.*, vol. 13, no. 5, pp. 289–303, 1977.

[7] L. J. T. Rodriguez, "A contribution to database flexible querying: Fuzzy quantified queries evaluation," Ph.D. dissertation, Simón Bolívar Univ., Caracas, Venezuela, Nov. 2005.

[8] R. Ebrahim, "Fuzzy logic programming," *Fuzzy Sets Syst.*, vol. 117, no. 2, pp. 215–230, Jan. 2001.

[9] M. Ishizuka and N. Kanai, "Prolog-elf incorporating fuzzy logic," in *Proc. 9th Int. Joint Conf. Artif. Intell.*, 1985, pp. 701–703.

[10] D. Li and D. Liu, *A Fuzzy Prolog Database System*. New York, NY, USA: Wiley, 1990.

[11] J. F. Baldwin, T. P. Martin, and B. W. Pilsworth, *Fril, Fuzzy and Evidential Reasoning in Artificial Intelligence*. New York, NY, USA: Wiley, 1995.

[12] P. J. Morcillo and G. Moreno, "Floper, a fuzzy logic programming environment for research," in *Proc. 8th Jornadas Sobre Programacion y Lenguajes (PROLE)*, 2008, pp. 259–263.

[13] F. Bobillo and U. Straccia, "FuzzyDL: An expressive fuzzy description logic reasoner," in *Proc. IEEE Int. Conf. Fuzzy Syst. (IEEE World Congr. Comput. Intell.)*, Jun. 2008, pp. 923–930.

[14] S. Guadarrama, S. Muñoz-Hernández, and C. Vaucheret, "Fuzzy prolog: A new approach using soft constraints propagation," *Fuzzy Sets Syst.*, vol. 144, no. 1, pp. 127–150, May 2004.

[15] C. Vaucheret, S. Guadarrama, S. Muñoz-Hernández, and LPAR, "Fuzzy prolog: A simple general implementation using CLP(R)," in *Logic for Programming, Artificial Intelligence, and Reasoning* (Lecture Notes in Artificial Intelligence), vol. 2514, M. Baaz and A. Voronkov, Eds. Berlin, Germany: Springer, 2002, pp. 450–464.

[16] S. Munoz-Hernandez, V. Pablos-Ceruelo, and H. Strass, "RFuzzy: Syntax, semantics and implementation details of a simple and expressive fuzzy tool over Prolog," *Inf. Sci.*, vol. 181, no. 10, pp. 1951–1970, May 2011.

[17] R. A. Ribeiro and A. M. Moreira, "Fuzzy query interface for a business database," *Int. J. Hum.-Comput. Stud.*, vol. 58, no. 4, pp. 363–391, Apr. 2003.

[18] P. Bosc and O. Pivert, "On a strengthening connective for flexible database querying," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ)*, Jun. 2011, pp. 1233–1238.

[19] G. Bordogna and G. Pasi, "A fuzzy query language with a linguistic hierarchical aggregator," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 1994, pp. 184–187, doi: 10.1145/326619.326693.

[20] J. Kacprzyk and S. Zadrozny, "SQLf and FQUERY for access," in *Proc. Joint 9th IFSA World Congr. 20th NAFIPS Int. Conf.*, vol. 4, 2001, pp. 2464–2469.

[21] G. Bordogna and G. Psaila, "Customizable flexible querying in classical relational databases," in *Handbook of Research on Fuzzy Information Processing in Databases*. Derry Township, PA, USA: Hershey, 2008, pp. 191–217.

[22] H. Prade, "Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries," *Inf. Sci.*, vol. 34, no. 2, pp. 115–143, Nov. 1984.

[23] M. Umano, I. Hatono, and H. Tamura, "Fuzzy databaase systems," in *Proc. IEEE Int. Joint Conf. Fuzzy Syst.*, vol. 5, 1995, pp. 35–36.

[24] J. M. Medina, O. Pons, and M. A. Vila, "Gefred: A generalized model of Fuzzy Relational Databases," *Inf. Sci.*, vol. 76, nos. 1–2, pp. 87–109, Jan. 1994.

[25] N. Konstantinou, M. C. Spanos, E. Solidakis, and N. Mitrou, "VisAVis: An approach to an intermediate layer between ontologies and relational database contents," in *Proc. CAISE 3rd Int. Workshop Web Inf. Syst. Modeling (WISM)*, 2006, p. 239.

[26] C. Martínez-Cruz, J. M. Noguera, and M. A. Vila, "Flexible queries on relational databases using fuzzy logic and ontologies," *Inf. Sci.*, vol. 366, pp. 150–164, Oct. 2016.

[27] Y. Takahashi, "A fuzzy query language for relational databases," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, no. 6, pp. 1576–1579, Nov. 1991.

[28] The CLIP Lab. *The Ciao Prolog Development System*. Accessed: Mar. 23, 2015. [Online]. Available: http://www.clip.dia.fi.upm.es/Software/Ciao

[29] J. Medina, M. Ojeda-Aciego, and P. Vojtáš, "A multi-adjoint approach to similarity-based unification," *Electron. Notes Theor. Comput. Sci.*, vol. 66, no. 5, pp. 70–85, Dec. 2002.

[30] J. Medina, M. Ojeda-Aciego, and P. Vojtáš, "A completeness theorem for multi-adjoint logic programming," in *Proc. 10th IEEE Int. Conf. Fuzzy Syst.*, Aug. 2005, pp. 1031–1034.

[31] J. Medina, M. Ojeda-Aciego, and P. Vojtáš, "Multi-adjoint logic programming with continous semantics," in *Proc. 6th Int. Conf. Logic Program. Nonmonotonic Reasoning (LPNMR)*. London, U.K.: Springer-Verlag, 2001, pp. 351–364. [Online]. Available: http://dl.acm.org/citation.cfm?id=646400.759097

[32] J. Medina, M. Ojeda-Aciego, and P. Vojtáš, "A procedural semantics for multi-adjoint logic programming," in *Progress in Artificial Intelligence* (Lecture Notes in Computer Science), vol. 2258, P. Brazdil and A. Jorge, Eds. Berlin, Germany: Springer, 2001, pp. 290–297.

[33] J. Medina, M. Ojeda-Aciego, and P. Vojtáš, "Similarity-based unification: A multi-adjoint approach," *Fuzzy Sets Syst.*, vol. 146, no. 1, pp. 43–62, Aug. 2004.

[34] J. M. Moreno and M. O. Aciego, "On first-order multi-adjoint logic programming," in *Proc. 11th Spanish Congr. Fuzzy Logic Technol.*, 2002, p. 16.

[35] V. Pablos-Ceruelo and S. Muñoz-Hernández, "Introducing priorities in rfuzzy: Syntax and semantics," in *Proc. 11th Int. Conf. Math. Methods Sci. Eng. (CMMSE)*, vol. 3, Benidorm, Spain, Jun. 2011, pp. 918–929.

[36] V. Pablos-Ceruelo and S. Muñoz-Hernández, "FleSe: A tool for posing flexible and expressive (fuzzy) queries to a regular database," in *Proc. 11th Int. Conf. Distrib. Comput. Artif. Intell.*, S. Omatu, H. Bersini, J. M. Corchado, S. Rodríguez, and P. Pawlewski, Eds. Cham, Switzerland: Springer, 2014, pp. 157–164.

[37] M. H. Deedar and S. Muñoz-Hernández, "Extending a flexible searching tool for multiple database formats," in *Emerging Trends in Electrical, Communications, and Information Technologies*. Singapore: Springer, 2018.

[38] M. H. Deedar and S. Muñoz-Hernández, "User-friendly interface for introducing fuzzy criteria into expressive searches," in *Intelligent Systems and Applications*. Cham, Switzerland: Springer, 2020, pp. 982–997.

[39] M. H. Deedar and S. Muñoz-Hernández, "Allowing users to create similarity relations for their flexible searches over databases," in *Artificial Intelligence and Soft Computing*. Cham, Switzerland: Springer, 2019, pp. 526–541.
[40] J.-B. Wang, Z.-Q. Xu, and N.-C. Wang, "A fuzzy logic with similarity," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 3, 2002, pp. 1178–1183.
[41] L. Godo and R. O. Rodríguez, "A fuzzy modal logic for similarity reasoning," in *Fuzzy Logic and Soft Computing*, K. Y. Cai, G. Chen, and M. Ying, Eds. Norwell, MA, USA: Kluwer, 1999.
[42] D. Dubois and H. Prade, "Comparison of two fuzzy set-based logics: Similarity logic and possibilistic logic," in *Proc. IEEE Int. Conf. Fuzzy Syst. Int. Joint Conf. 4th IEEE Int. Conf. Fuzzy Syst. 2nd Int. Fuzzy Eng. Symp.*, vol. 3, Nov. 2002, pp. 1219–1226.
[43] F. Esteva, P. Garcia, L. Godo, E. Ruspini, and L. Valverde, "On similarity logic and the generalized modus ponens," in *Proc. IEEE 3rd Int. Fuzzy Syst. Conf.*, vol. 2, Dec. 2002, pp. 1423–1427.

**Mohammad Halim Deedar** received the B.C.A. degree in computer applications and the M.Sc. degree in computer science from Jamia Hamdard University, New Delhi, India, in 2014 and 2016, respectively. He is currently pursuing the Ph.D. degree in software systems and computing with the Polytechnic University of Madrid, Madrid, Spain.

His current research interests include data management and analytics, fuzzy logic, logic programming, and machine learning.

**Susana Muñoz-Hernández** received the B.S. degree in licenciada en informática from the Society of International Studies of Madrid, Madrid, Spain, in 1997, the M.S. degree in management of information technologies from the Ramón Llull University, Barcelona, Spain, in 2003, and the Ph.D. degree in computer science from the Polytechnic University of Madrid (UPM), Madrid, in 2003.

She has professional experience in some companies (the International Conference on Telecommunications (ICT) and bank sector) and joint research (national and European) projects of recognized prestige. She has been an Associate Professor with the Computer Science School, UPM, since 1998, where she develops her research activity with the BABEL Group, with more than eighty publications. She set up the Technology for the Development and the Cooperation Group, Madrid, in 2006, where she is currently the Director. She has directed various projects oriented to improve education in developing countries, including Burundi, Ethiopia, Kenya, and El Salvador. She is currently the Director of the Educational Innovation Group—Technology Innovation for Educational Development, Madrid. Her teaching experience is in computer programmings, such as methodology of programming, logic and functional languages, constraint programming, and fuzzy logic, free software applications, and personal skills, such as communication, relation, presentations in public, negotiation, conflict solving, and management of intercultural teams.

Dr. Muñoz-Hernández was a member of the Advisory Board of the Direction of Cooperation for the Development of UPM International Relations and the Advisory Board of the Conference of Rectors of Spanish Universities, University Observatory of Cooperation for Development. She was a member of the Manager Committee of the Spanish Platform for Software and Services from 2008 to 2010. She is a member of the Management Board of the itdUPM, Center of Innovation in Technology for Human Development. She received the first prize in a national competition for talented young people organized by the University of La Salle, Madrid, in 2003. She received the 2011 UPM prize of research in international cooperation for development. She coordinated the European Master in Computational Logic (first official Erasmus Mundus master in computer science) from 2004 to 2008. She is part of the committees of several international conferences in her area, including the ACM Symposium on Applied Computing (SAC), the Spanish Conference on Programming and Computer Languages (PROLE), the International Conference on Information Processing and Management of Uncertainty (IPMU), the Workshop in Artificial Intelligence applied to Mobile Robotics (WCAFR), the International Work Conference on Artificial and Natural Neural Networks (IWANN), SERVICE COMPUTATION, EDUCON, the IEEE Education Society's flagship Asia-Pacific Conference Series (TALE), Fuzzy Computation Theory and Applications (FCTA) en the International Joint Conference on Computational Intelligence (IJCCI), WomENcourage, ACM Symposium on Computing and Development (DEV), the European Conference on Sustainability, Energy & the Environment (ECSEE), and FUTURE COMPUTING. http://babel.ls.fi.upm.es/ susana/publications.html