Adaptative and personalized web framework for searching data using flexible queries

Victor Pablos Ceruelo[1], Dan Seeruttun - - Marie[1] and Susana Muñoz Hernandez[2]

[1]*PHD in ..., Universidad Politecnica de Madrid, My Street, MyTown, MyCountry*

[2]*Student in Software Engineering, Universidad Politecnica de Madrid, Rennes, France*

[3]*Computer Science..., Universidad Politecnica de Madrid, MySecondTown, MyCountry*

{*victorpablosceruelos, dan.seeruttun*}*@gmail.com, susana@fi.upm.es*

Abstract:

The abstract should summarize the contents of the paper and should contain at least 70 and at most 200 words. The text must be set to 9-point font size.

1 INTRODUCTION

When trying to model a problem, assigning each concept to a term and an universal meaning is very important. These names must be very clear for anyone. However, problems can appear during the modelization of a fuzzy or subjective problem, where terms meaning vary and are not clear.

For instance, trying to model the concept of the predicate expensive car is difficult because it involves each person own idea of expensive to represent it. Secondly, how can a car be defined with an expensive and non expensive term ? If sticking to the reality is an objective, being able to define the expensiveness of any product is the key.

Thus a better representation of the predicate is needed, associating a cost and its match with the predicate is the idea. That's introducing the concept of truth value, seen in (Citation Victor Thesis), a value that will represent the match between fuzzy values and their predicates, or in our example costs and the predicate expensive. With this modelization, it will become possible to search through databases, coupling truth values and fuzzy predicates and even going further by personalizing predicate to users. In this paper, FLESE will be presented, an implementation of the fuzzy concept through databases, adaptative and able to personalize to each user.

2 FLESE

Generalities FleSe is an user friendly interface for the search engine provided by Rfuzzy, with the capability to evaluate the (possibly) fuzzy query against a database containing no fuzzy information. By using it we can answer queries like "I want a restaurant close to the center serving Mediterranean food or similar" from a database that contains a list of restaurants with the distance of each one to the page 108. (Compilation date: April 19, 2015) CHAPTER 5. REAL APPLICATION CASES center (100 meters, 1000 meters, etc) and the food they serve (Mediterranean, Spanish, Italian, etc) and some meta-information linking fuzzy and non-fuzzy information. Technical information

Flese is composed of an engine, an improved version of RFuzzy: Syntax, semantics and implementation details of a simple and expressive fuzzy tool over Prolog". In this new framework we have included the management of quantifiers (even negation), similarity, overload of attribute's names and others. Its main advantage over some other engines is that its syntax is trivial, it can use both fuzzy and crisp data, it allows to reuse existing databases and Prolog code and it has every facility we need to represent real-world applications. Rfuzzy is a package of the Ciao Prolog logic programming environment. The web interface is a

Java application that interprets the answers provided by the fuzzy framework. It runs on a Tomcat server behind an Apache proxy. FleSe core is developed as a Ciao Prolog package, while the web interface is managed by a Java application running on an Apache Tomcat Debian (Linux) server. The client part of the web interface is developed in HTML and JavaScript and uses Ajax to improve the usability. Besides, we use some libraries developed by others. Mainly: SocialAuth, OpenId4Java, jQuery, jQuery UI and Highcharts JS. The architecture is the following: The users connects himself to Flese's interface. with an Internet Browser. FleSe's web interface is an application written Java that runs on a Tomcat server, on a machine with a Linux operating system. The Tomcat server listens to http(s) requests and, when it receives one for the FleSe application, it redirects it to the Servlet in the FleSe's application in charge of processing and answering it.In order tonprocess and answer the request the Servlet evaluates what it is asking for and decides whether it needs to call the core for answering it or not. When we press the search button most of the times it calls the core, but if the query has been posed recently it will use cached results to decrease the response time. The core is RFuzzy, a Prolog program that reads a prolog configuration file to determine to which database connect to (and how), link multiple tables into a virtual database table or get rid of columns with not needed information, give meaning to each column in the virtual database table and add concepts (fuzzy or not) to the list of concepts with which the user can build queries. . Once the configuration file is chosen by the user the web interface asks the core to load it.

To increase the speed of answering a query, the most common queries are pre-cached. In others words, the most repetitive data and its similar queries are kept in the cache. Fuzzy processing The system has got semantic recognition, it can recognize the queries it can process and orient the user with its interface, or it can refuse if it is not possible. For this semantic recognition, the file owner has to implement what are the words used in queries, defining them with direct access to the database or combining data from it: close to the center Combining easily predicates make the file owner free to implement whatever he wants. He can as well implement similarities between predicates. If the user wants to access a Mediterranean restaurant, there is no easy decomposition of what is this kind of restaurant, but it can be

defined combining other predicates with similarities. The data is computed by the core, and the solutions of the query are displayed on Flese, ordered by their truth value, a value between 0 and 1 that determines the validity of the solution.
Example of use Each user has to identify himself to Flese via a Social network provider, and then he gets access to the application. The user can add his configuration file, or browse the one already existing. One of the advantages of Flese is it can work with more than one database for the queries, and you can have access not only to your file, but to the files other users shared. To make a query, first the user has to choose the subject of his query, and then the predicates available are displaying, fuzzy and crisp, using the core of Rfuzzy. Some of the queries have more data to insert than others, it will be displayed automatically. Modifiers are as well available, like negation not, minimum min, and quantity modifiers are accessible too like very, or too much. Finally, for the numerics predicates, it is possible to use comparisons operators to complete more complex queries. Finally, the results is displayed on a table, provided and ordered with the truth value.

3 ADAPTATIVE PERSONALIZED FRAMEWORK

3.1 USER PERSONALIZATION

Flese application brings the treatment of the fuzzy queries but it does not solve the problem of the subjectiveness of the fuzzy queries. Indeed, an expensive car does not have the same meaning for everyone.

The point of the user personalization is to give to the user the opportunity to modify the definition of the fuzzy predicates, thus to overcome the subjectivity of the vocabulary used. Moreover, a personalization does not affect other users, so each user can create his own definitions of the same predicate to access the same database using Flese. The database file will then contain all users personalization.

To personalize a predicate, the predicate is divided in sections that can be personalized by the user. Consequently, not all the predicate can be personalized. For example, let the predicate

"height of car in mm". This predicate is non fuzzy and it is impossible to personalize it. Furthermore, the owner of the file can decide or not to let users personalize a predicate.

To make a predicate personalizable, the owner of the file has to divide the predicate in sections. The user will be able to assign values to each sections. For instance, using the predicate expensive, these could be the different sections to take into account.

- From 0 € to 10000 €
- From 10000 € to 30000 €
- From 30000 € to 100000 €
- More than 100000 €

At each section will be associated a truth value, and this value is the one the user will be able to edit.

This algorithm is the first default function for the predicate price_in_euros applied to car. The first value is the price and the second is the truth value.

3.2 IMPLEMENTATION ON FLESE

The system of sections has been implemented on Flese. To authorize the modification of the file, the prolog syntax of the file for the predicate expensive is the following:

```
expensive(car) :~
function(price_in_euros(car), [ (0, 0),
(10000, 0.5), (30000, 1), (100000, 1) ]).
```

Let's demonstrate the usefulness of this fuzzy predicate personalization.
We choose a user that is looking for an expensive car. If he runs the query directly, he will find the following solutions in the table 1:

Table 1: Results of simple expensive car query

N0	Name	Price	Truth value
1	VW Caddy	45000	1
2	Alfa Romeo	30000	1
3	Aston Martin	150000	1
4	Ford S Maxi	30000	1
5	Audi TT	40000	1
6	Audi Quattro	48000	1

The 6 first results will be expensive car with a truth value of 1, this table of results will not

please a rich user, that will be interested in a higher cost for the predicate expensive.

Then he will assign a truth value to different ranges of prices, from inferior to 10000 € to more than 100000 €. A rich user may personalize this predicate imposing that a car valued at 30000 € is not really expensive (truth value = 0.40) and a car valued at 100000 € is very expensive (truth value = 1). After being modified, a message appears confirming the modification. The figure 1 illustrates this example.

Figure 1: Fuzzification personalization.

After having modified the predicate, the user can still have a look to the default value of the predicate in case of he is lost. This part remains important to help the user to get back from a wrong personalization. The result of the query is presented in the table 2:

Table 2: Results of fuzzy expensive car query

N0	Name	Price	Truth value
1	Aston Martin	150000	1
2	VW Caddy	45000	0.55
3	Audi TT	40000	0.49
4	Audi Quattro	48000	0.41
5	Alfa Romeo	30000	0.4
6	Ford S Maxi	30000	0.4

Asking for his expensive car, the user knows by reading the truth values that there is only one expensive car according to his definition. The user is taking into account only the cars with truth values around 1. The advantage of the truth values is the user can easily select the expensive cars with a 60
The owner of the file can decide which predicate he wants to authorize the personalization. He

keeps the control on the file, he can impose a default value of the fuzzification before any personalization.

3.3 MACHINE LEARNING FOR IMPROVING SEARCHING CRITERIA

Starting considering an increasing number of users, it becomes interesting to search for better predicate criteria. Indeed, the users joining the application after the others can benefit from the others data. Knowing that people share some predicate personalization can lead to an improvement of the criteria for the newcomers. In that sense, Flese changes the values of the default function for the newcomers' and non personalized's predicate according to what is most representative to the users.

Knowing that the predicate are divided in sections that are independant from each others, the algorithm is applied on each of these sections individually.

The median has been chosen for this algorithm to fit with the higher number of people's criteria, so the new user won't have to personalize the fuzzification, or only parts of it. Moreover, the median is better than a simple average because the value will fit an actual personalization. It become important on the case we have 2 opposites types of population. The average algorithm will give us an average value, that will fit none of the 2 types of the population.

Meanwhile the median algorithm will give us the personalization of one user belonging to the biggest population. That is why the median algorithm is better.

Theses changes will have no effect on the parametrized fuzzification if it has been already changed. For example, given a population A and a population B having different behaviours with the predicate expensive, if we introduce more people of population A than people of population B, the definition of the predicate will adapt to the situation, following the population A predicates.

The searching criteria will then adapt to the population of the users. Even after deciding to invert the population rate, by adding more poor people, the predicate criteria will be personalized by the new population to fit their idea. In that case, the criteria will represent more the poor people.

For instance, if most of the people consider that 10000 € and 30000 € is not expensive, the default expensive predicate will adapt according to it. Here is an example showing the adaptation in figure 2:
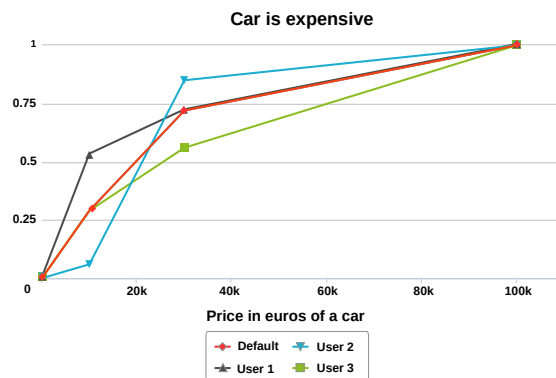

Figure 2: Default function determined by active learning.

In this example, 3 users personalized the predicate expensive. The Default function follows the median algorithm for each section.

For 10000 € it is following the user3, and for 30000 € it is following the user 1. The point of this figure is to show that the Default function is able to fit with most of the population of users.

4 CONCLUSIONS

TODO

REFERENCES

APPENDIX

If any, the appendix should appear directly after the references without numbering, and not on a new page. To do so please use the following command: $\backslash section*\{APPENDIX\}$