A Fuzzy Approach to Faceted Classification and Retrieval of Reusable Software Components

Francesco Baruchelli and Giancarlo Succi
DISA, Università di Trento, Via Inama 5, I-38100 Trento, Italia
e-mail: Francesco.Baruchelli@lii.unitt.it, Giancarlo.Succi@cs.unitt.it

Abstract

Software reuse can be very useful in increasing the productivity and the quality level of a company, but appropriate classification and retrieval tools have to be provided in order to exploit its pros. The classic classification and retrieval methods based upon a controlled, fixed vocabulary are not very flexible and can be unsatisfactory in many cases, especially when the specification are not fully defined. An improvement in this sense can be obtained applying some simple fuzzy concepts to the well known faceted classification method. Such a method based upon the same structures used in the traditional faceted classification is described. The method allows the user to specify how much a keyword is representative of the searched components, giving the means to specify the undesired keywords, too, through a set of fuzzy attributes.

1. Introduction

The adoption of a good reuse policy can bring many pros to a software company. Apart from the obvious gain in productivity, the quality of the resulting systems and their easiness of maintenance can be improved, too. Nonetheless, a series of problems which arise at different levels make it difficult to reach these goals. One of the simplest and most obvious of these problems is the difficulty to find the right component to reuse in a determinate context. Many software libraries have been constructed using and combining different methods for the classification and the retrieval of the reusable entities, such as a free text description, or a classification using a set of keywords. A general problem of the majority of these methods is the fact that the result of the search consists only of the components which have a descriptor which perfectly matches the query, and the query is not easily releasable in an efficient way. This can be avoided through the use of the faceted classification [5] which returns as the result of a search the set of the elements with the description which is the "nearest" to the one of the searched entity. A drawback of this method is the fact that it is based on the use of a controlled vocabulary that is composed by a finite number of fixed terms. This means that the user is not always able to describe exactly what he is looking for using only this fixed set of keywords. Furthermore there is not the possibility of specifying some negative conditions on the search, i.e. the user can not look for something that is not described by a certain keyword.

In this paper we present a fuzzy version of the faceted classification and retrieval method which allows to overcome in part the above mentioned drawbacks of this technique.

In Section 2 we present the classification scheme for the reusable components, while in Section 3 the search engine based on this information is described.

2. The faceted Classification with a Fuzzy Approach

The faceted classification is a sort of refinement and improvement of the keyword classification. The difference between the two methodologies is the fact that here the keywords do not belong to a plain world, but are divided in subsets which describe different views, or facets, of the classified entities. Furthermore, the facets themselves are not plain sets of keywords, but are organized in a weighted tree structure representing generalization and specification relationships between the keywords. Figure 1 represents a sample of the structure of a faceted dictionary of keywords, where there are two facets with a natural meaning: Abstraction and Operates On. Here the numbers associated to the edges represents the distance between the keywords with 0 for the maximum distance, and 1 for the minimum one.

In the traditional approach to this kind of classification and retrieval method, a keyword for each of the facets present in the vocabulary is associated to every component in the library. Then in the retrieval phase the tree structure and the weights associated to the edges are used to estimate a distance between the keyword associated to the existent components and the keywords in the query describing the desired entity. Finally the distances on each of the facets are combined to give the distance between each of the existing components and the entity described by the query. These values are used to rank the reusable components and only the nearest ones are returned to the user.

The query can be easily released presenting in the result some of the components which are at a greater distance from the perfect entity. As we will see in the next Section, we use the tree and the weights in a different, fuzzy, way to perform the retrieval of the reusable components, while the classification phase is very similar to the traditional one. In fact, we still associate the most representative keyword of each of the facets to the components, but in addition we add a numeric value which can vary in the range [0,1] to express the degree of suitability of the keyword to the description of the considered component.

![Diagram of a Facet Dictionary]

**Figure 1: Sample of a Facet Dictionary**

In this way a new degree of freedom is introduced, allowing a better representation of the components with the limited number of terms belonging to the vocabulary.

### 3. The Search Engine

First of all we have to define a format for the queries. Since the keywords stored in the facets do not always represent exactly the component the user is looking for, a degree of fuzziness is introduced. For each of the facets the user can specifies one or more keywords and state the degree of evidence with which each of these keywords represents the desired software entity. This is done associating one of the following four specified degrees to the keywords, ordered from the most to the less representative:

- VH: very high;
- H: high;
- L: low;
- VL: very low.

This allows the user to deal with incomplete or imprecise specifications, and introduces negative statements in the query (the user can impose that a determinate keyword must represent the desired component with a very low degree). Of course the result of the query on each of the components stored in the library will not be a binary value, but will vary in the range [0,1] depending on the degrees associated to the keywords in the query and the values associated to the keywords in the components' descriptors. The pseudo-code describing the search engine can be found in Figure 2. It simply calculates a degree of satisfaction for the query for each of the components in the library and returns as output all the components with a value for the degree of satisfaction equal to this maximum. The computation is done dividing the query in many sub-queries related to the single facets.
The final value of the degree of satisfaction of a component for the query is obtained as the sum of the degrees of satisfaction for all of these sub-queries.

### 3.1 Computation of the Degree of Satisfaction of the Sub-Query related to a Single Facet

The sub-query related to a facet can be expressed in the following form: \( k_1 = d_1, k_2 = d_2, \ldots, k_n = d_n \) where \( k_i \) is a keyword in the considered facet, and \( d_i \) is the degree of satisfaction wanted for the keyword \( k_i \) for the artifacts satisfying the query.

The degree of the satisfaction \( dFC \) of a single component for the sub-query can be expressed solving the following fuzzy expression:

\[
\text{IF AND } k_1 = d_1 \text{ AND } k_2 = d_2 \text{ AND } \ldots \text{ AND } k_n = d_n \text{ THEN } dFC
\]

where the fuzzy value of \( dFC \) is the minimum of the fuzzy value obtained evaluating the conditions \( k_i = d_i \) (dQKFC in Figure 2).

```plaintext
maxdC=0; // maximum degree of satisfaction
1C=empty; // list of all the components with the // maximum degree of satisfaction
for each component C in the library {
    dc=0; // degree of satisfaction of C for
    // the query Q
    for each facet F in the vocabulary {
        dFC=1; // degree of satisfaction of C for the // part of Q related to F
        for each keyword K in Q belonging to F {
            using the keyword related to F in the descriptor of C and the tree structure of the Facet,
            calculate the degree dKFC with which K represents C;
            starting from dKFC, obtain dQKFC representing the degree of satisfaction of C for the part of Q involving K;
            if dQKFC < dFC then
                dFC = dQKFC;
        }
        dc=dc+dFC;
    }
    if dc > maxdC then {
        if empty 1C then {
            empty 1C;
            maxdC=dc;
        }
        add C to 1C
    }
}
return 1C and maxdC as output;
```

**Figure 2: Pseudo-Code for the Search Engine**

These values are obtained starting from the values representing the degree of representation of \( k_i \) for the component (dKFC), passing it as the argument to an appropriate function. There is such a function for each of the four degree of satisfaction allowed in the query. Of course if \( k_i \) is scarcely representative of \( C \), the value of \( k_i = VH \) will be next to 0, while the value of \( k_i = VL \) will be next to 1. The functions selected for this purpose can be found in Figure 3.

![Fuzzy Functions](image)

**Figure 3: Fuzzy Functions**

Each function can be represented by means of three numerical intervals representing the range of values of dKFC in which the function grows linearly from 0 to 1, the range in which the value of the function is 1, and the one where the function decreases, again in a linear way, from 1 to 0. The value of the function is 0 for each dKFC outside these intervals. With this representations the functions can be described by Table 1.
Table 1: Fuzzy Functions

In order to complete the computation of the degree of satisfaction of the sub-query, we need to determinate the value of dKFC, i.e., the degree of representation of the keyword $k_q$ for the considered component.

Let $k_q$ be the keyword appearing in the query, $k_c$ the keyword of the descriptor of the component for the facet to which $k_q$ belongs, and $d_c$ the value associated to $k_c$ in the descriptor.

There are four different cases that have to be considered for the computation of dKFC:

- $k_c = k_q$;
- $k_c$ is an ancestor of $k_q$ in the tree representing the facet;
- $k_c$ is an ancestor of $k_q$ in the tree representing the facet;
- $k_c$ and $k_q$ belong to the same tree (they are keywords of the same facet), so they have a common ancestor, but neither one is a direct ancestor of the other.

The first case is the simplest, since $d_c$ represent the wanted degree and we can simply write: $d_{KFC} = d_c$.

If $k_c$ is an ancestor of $k_q$, then $k_c$ is a specialization of $k_q$, and we can write again: $d_{KFC} = d_c$.

For instance, let us consider the sample facet of Figure 4: if we are looking for a "pen", and we find an object which is described by the keyword “blue pen” with $d_c=0.7$, we can say that the object is described by the keyword pen with $d_{KFC}=0.7$.

![Figure 4: Sample Facet](image)

3. Things are not so simple if $k_c$ is an ancestor of $k_q$. In this situation, we are looking for something that is more specific than the found component. This means that we have to find a sort of probability of the component being the right specification of the found keyword. This can be done exploiting the tree structure and its associated weights.

If $k_c$ is the father of $k_q$, then we can calculate this probability $P$ as the weight of the edge linking $k_q$ to $k_c$, divided by the sum of the weights of all the edges starting from $k_c$. Then we can compute $d_{KFC}$ as follows: $d_{KFC} = d_c \cdot P$. For instance, referring again to Figure 4 for the tree, if we are looking for a “dark pen” and we find something which is a pen with $d_c=0.6$, we have $P=0.8/(0.8+0.4)=2/3$, and $d_{KFC}=0.6 \cdot (2/3)=0.4$. If $k_c$ is not the father of $k_q$, but only one of its ancestors, the computation is very similar and is done in a recursive way. If we denote with $k_i$ the intermediate keywords in the path leading from $k_c$ to $k_q$ (with $k_c = k_0$ and $k_q = k_n$) for each of the involved edges a probability $P_i$ can be computed in exactly the same way seen above. The total $P$ will then be the product:

$$P = \prod_{i=1}^{n} P_i$$

And once again we write:
\[ d_{KFC} = d_c \times P = d_c \times \prod_{i=1}^{n} P_i \]

For instance, if we are looking for a "black pen" and the considered component is a "pen" with a degree of 0.7, we have:

\[ P_c = 0.8/(0.8+0.8) = 2/3, \quad P_e = 0.5/(0.8+0.6) = 0.6/1.4, \]

leading to \( P = (2/3) \times (0.6/1.4) = 0.2/0.7 \). Finally, we can compute the value of \( d_{KFC} \):

\[ d_{KFC} = 0.7 \times (0.2/0.7) = 0.2. \]

4. Finally, if \( k_c \) and \( k_q \) have an ancestor in common, but they are not one a direct ancestor of the other one, a combined version of the cases 2 and 3 is applied to compute \( P \), then \( d_{KFC} \) is again obtained as the product of \( d_c \) for \( P \). To compute \( P \), the first common ancestor \( k_c \) of \( k_c \) and \( k_q \) is found. This keyword can be considered as \( k_c \) with respect to \( k_q \), and vice-versa.

When \( k_c \) is considered as \( k_c \), the situation is the same as in case 3, and \( P_c \) is computed recursively as above, the only difference is the fact that here the keywords \( k_c \) are the ones appearing in the path leading from the first ancestor of both \( k \) and \( k_q \) to \( k_q \).

When \( k_q \) is considered as \( k_q \), the situation is very similar to what we have seen in case 2, and therefore \( P_q \) is set to 1.

At this point we compute the total \( P \) as the product:

\[ P = P_c \times P_q \]

And once again we write: \( d_{KFC} = d_c \times P \) For instance, if we are looking for a "blue pen" and the considered component is described by the keyword "red pen" with degree 0.5, we have:

\[ P = (0.8/(0.8+0.4)) \times (0.8/(0.8+0.6)) = 0.8/2.1 \]

and \( d_{KFC} = 0.5 \times (0.8/2.1) = 0.4/2.1 = 0.19. \)

4. An Example of Application

We will apply the search engine to a simple example with the single facet of Figure 4, and two elements with the descriptors shown in Figure 5.

![Figure 5: Objects' Descriptors](image)

Let us suppose that we are interested in finding a pen which we want to be dark, and we would prefer to be blue. This can be expressed with the following query: dark pen=VH, blue pen=H

If we consider Object1, we have first of all to determine the degree of satisfaction of each of the keyword in the query. The keyword in the descriptor is "Black Pen", this means that when we consider the keyword "Dark Pen" we fall in case 2, and \( d_{KFC} = 1 \). When we consider "Blue Pen", we fall in case 4 and the computation gives \( d_{KFC} = 1 \times 0.8/(0.8+0.6) = 0.8/1.5 = 0.53. \)

Passing the values of \( d_{KFC} \) to the fuzzy functions, we can compute the degree of satisfaction for the sub-queries, with the result:

- dark pen=VH 
  VH(1)=1;
- blue pen=H 
  H(0.53)=0.65;

So the degree of satisfaction of Object1 for the query is the minimum of the above values: \( d_C = 0.65. \)

The same procedure applied to Object2, which is represented by the keyword HB with degree 0.7, gives the following values for \( d_{KFC} \):

- dark pen=0.7 \times (0.8/(0.8+0.7)) \times (0.8/(0.8+0.4)) = 0.25
- blue pen=0.7 \times (0.8/(0.8+0.7)) \times (0.8/(0.8+0.4)) \times (0.8/(0.8+0.6)) = 0.14

This data lead to the following results for the values of \( d_{QKFC} \):

- dark pen=VH 
  VH(0.25)=0;
- blue pen=H 
  H(0.14)=0;

So the degree of satisfaction of Object1 for the query is the minimum of the above values: \( d_C = 0. \)

The comparison of the values of \( d_C \) for the two objects forces the search engine to correctly return Object1 as its result.

5. Conclusions and Further Work

Finding the software component with the greatest possibilities of successful reuse is a hard task, especially when the characteristics of the wanted artifacts are not completely specified. In this paper we have proposed a different approach to the classical faceted classification and retrieval methodology which try to increase its flexibility, and to expand the range of queries it allows. This has been done first of all introducing a parameter in the descriptor describing the degree with which a keyword describes a component. The queries have been refined, too, allowing the user to specify if a keyword represents the characteristics of the searched artifact at a very high, high, low, or very low level.
The search algorithm is not too different from the classical one used in the faceted approach. In fact, it makes use of the tree structures of the facets and the weights associated with their edges. The use of the same structures allows the coexistence of the new search engine with another one implementing the usual algorithm. A limited series of tests has been conducted, leading to results that look sound, and confirming an increased degree of flexibility. It has to be noted that the results are highly dependent from the proper classification of the components, and that a fundamental role is played by the person who creates the facets' trees and assign the weights to the edges. A bad work in this phase can make useless all the methodology, but this is a characteristic that is proper of the classical faceted classification, too.

The system is currently been integrated in Sarto, a library of reusable software entities which implements some classical search methodology, including the faceted one. After the system will be completed, a more complete set of tests will be performed, and all the methodology will be refined, especially looking for more complex fuzzy functions and tuning the way the weights of the trees are combined.

References