A GENERAL FRAMEWORK FOR COMPUTING WITH WORDS IN OBJECT-ORIENTED PROGRAMMING

F. BERZAL*, J. C. CUBERO†, N. MARÍN‡ and M. A. VILA§

Department of Computer Science and Artificial Intelligence,
University of Granada, 18071, Granada, Spain
*fberezal@decsai.ugr.es
†JC.Cubero@decsai.ugr.es
‡nicm@decsai.ugr.es
§vila@decsai.ugr.es

J. KACPRZYK¶ and S. ZADROŻNY‖

Systems Research Institute, Pol. Acad. Sci.,
ul. Newelska 6, 01-447 Warsaw, Poland
¶kacprzyk@ibspan.waw.pl
‖zadrozny@ibspan.waw.pl

Computing with words (CWW) techniques have been shown to be useful in the management of imperfect information. From the programmer’s standpoint, new tools are necessary to ease the use of these techniques within current programming platforms. This paper presents a step in this direction by describing a general framework that supports the implementation of applications dealing with fuzzy objects. We pay special attention to the study of the object comparison problem by offering both a theoretical analysis and a simple and transparent way to use our theoretical results in practice.

Keywords: Computing with words, soft computing, object-oriented programming, fuzziness.

1. Introduction

Computer science has experienced a brilliant evolution till the advent of the so-called “Information Society”. This evolution, firstly motivated by a need for more and more effective and efficient computation capabilities, is now also guided by a desire to bring computers closer to the human way of communication and reasoning. For a human being the only fully natural means of communication and articulation is natural language. Therefore, to attain that human consistency, computations should proceed not only on numbers but also on words.

Both Zadeh’s fuzzy set theory and his attempt to develop a computational theory of perceptions represent some of main advances to meet this challenge. Many researchers have successfully used these theories to represent information
with vagueness that natural language usually involves, and reasoning schemes based on such representations. Techniques of computing with words (and perceptions)\textsuperscript{21,22} have proved to be a suitable solution for the management of imperfect information. And these techniques need new tools to make their use easier within current programming platforms.

During the last decade some concepts of computing with words have been used to extend data models in order to allow the representation of imperfect data. Proposals can be found for the relational,\textsuperscript{12} object-oriented\textsuperscript{8} and object-relational\textsuperscript{5} database models. Nevertheless, programmers still have no assistance when they need to develop applications dealing with imperfect data. Easy-to-use transparent mechanisms to cope with this kind of applications are therefore of great interest and relevance.

Programming techniques have made significant advances during the last years. Conventional procedural programming has been progressively substituted by object-oriented programming to the extent that most computer applications currently under development follow this paradigm. The ability to describe a program as a collection of objects interacting among themselves provides an anthropomorphic environment suitable for large development efforts.

Many programming platforms have appeared to develop object-oriented code. Among them, two products are clear market leaders: Sun Microsystems’s Java Technology (http://java.sun.com) and Microsoft’s .NET Framework (http://msdn.microsoft.com/netframework/).

In Ref. 10 a framework is presented which allows programmers to handle imprecision in the description of objects and responding to the aforementioned need for facilitating the incorporation of soft computing techniques in modern programming platforms. The framework has been implemented for both Java\textsuperscript{1} and C\textsuperscript{2}.

In this paper we present a generalization of the C\textsuperscript{2} framework which takes advantage of the advanced features the .NET platform incorporates. This generalization improves the way programmers can develop their applications by allowing them to adapt the framework behavior to the particular semantics of the problems being solved.

The paper is organized as follows. Section 2 introduces the problem of fuzzy object representation in conventional object-oriented platforms by describing the issue of a complex object comparison and presenting a general approach to compute an object compatibility (degree). Section 3 includes a comprehensive overview of operators which can be selectively chosen to adapt the general approach to the semantics of particular problems. Section 4 is devoted to study the problem of cycles in the data graph. Section 5 describes our class framework and illustrates its use by means of an example in C\textsuperscript{2}. Finally, some conclusions close the paper.
2. Enabling the Implementation of CWW Applications

Improvements in the representation and management of objects which can be fuzzily described are desirable in modern programming environments. To attain this goal we need both a variety of new structures to handle the representation of imprecise data (built according to different semantic interpretations of imprecise values), and a code for performing data comparison (written using suitable operators for fuzzy data handling).

Moreover, we should avoid the need to write a sophisticated code to deal with fuzzy objects in order to promote a wider use of our approach. In fact, our solution to this problem is based on the concept of reuse and on a desire to simplify the use of our framework to programmers developing their own applications.

2.1. Fuzzily described objects

The state of an object is equated with a set attribute values according to its class description. In order to give a support for the representation of fuzzily described objects, we have to consider different types of attribute values. As well as the conventional precise values, objects, and crisp collections of the conventional object-oriented data model, we have to allow programmers to use more powerful values as imprecise labels and fuzzy collections in order to deal with imprecision in the state of the object.

Figure 1 shows a description of different types of attribute values that we consider in our approach.

As can be seen in the figure, the description of the object’s state can be composed by:

- Precise values: This category of values involves all the classical basic classes that usually appear in an object-oriented data model (e.g. numerical classes, string classes, etc.). Values of this kind of domains are easily represented and compared using conventional built-in data types and the classical set of relational operators.
- Imprecise values: The case of imprecise values is a bit more complex. Linguistic labels\textsuperscript{17,18,19} are usually used to express this kind of values. Different types of imprecise values must be considered according to the semantics of the imprecise value. As we will see, comparisons are performed via a generalization of the equality concept. This generalization must be developed according to the domain nature.
- Objects: the attribute value may be a reference to another object (constituting what is called a complex object). If objects are fuzzily described, they also need for a generalization of the classical value equality in order to be compared.
- Collections: the attribute may be conformed by a set of values or, even, by a set of objects. Imprecision in this kind of attributes appears at two levels:
  - The set may be fuzzy.
  - The elements of the set may be fuzzy values or, in general, fuzzily described objects.
The comparison of collections will need for special operators which take into account their conjunctive semantics.

### 2.2. Generalization of equality

According to the previous discussion, we do not only need a representation of fuzzily described objects in our programs but also a way to manage this kind of objects. This includes a basic capability to compare the state of two objects belonging to a given class.

In the above mentioned imprecise context, this comparison has to be performed by a process which consists of two steps:

- computing degrees of compatibility for pairs of attribute values, and
- aggregating these compatibility degrees to obtain a general degree of compatibility of the objects.

That is, if \( o_1 \) and \( o_2 \) are two objects belonging to class \( C \) which is characterized by type \( T_C \) whose structural component \( Str_C \) is an attribute set \( \{a_1, a_2, \ldots, a_n\} \),

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precise values</td>
<td>Classical basic classes that usually appear in an object-oriented data model.</td>
</tr>
<tr>
<td>Imprecise values</td>
<td>Different types of linguistic labels, according to the semantics of the imprecise value.</td>
</tr>
<tr>
<td>Objects</td>
<td>References to other objects (complex objects).</td>
</tr>
<tr>
<td>Collections</td>
<td>Sets of objects. These sets may be fuzzy and/or the elements of the sets may be fuzzy values (even fuzzily described objects).</td>
</tr>
</tbody>
</table>
then our goal is to find a compatibility degree between $o_1$ and $o_2$ by the aggregation of compatibility degrees of the attributes’ values $\{o_i, a_1, o_i, a_2, \ldots, o_i, a_n\}$.

$S_{a_i}(o_1, o_2)$ will stand for the compatibility degree observed between the $a_i$ attribute values for objects $o_1$ and $o_2$, and $S(o_1, o_2)$ will be an aggregated compatibility degree we want to calculate.

It should be noted that, in this process, not all attributes need to have the same importance. For example, if we are comparing people, some attributes may be more discriminating (e.g., name, father’s name) while others may be less discriminating (e.g., weight, age). To reflect this fact, each attribute $a_i$ has an associated weight $p_{a_i} \in [0, 1]$ that represents its relative importance in the final decision. Moreover, the assignment of importance may be influenced by the purpose of comparison.

Figure 2 summarizes the process of calculating compatibility degrees. In order to compare objects $o_1$ and $o_2$, we first compare their attribute values and obtain partial compatibility degrees $S_{a_i}(o_1, o_2)$. Then, we aggregate them to obtain a global compatibility opinion $S(o_1, o_2)$ according to the attribute importance set in the class.
2.3. Aggregation of compatibility values

The computation of a compatibility degree between two objects of the same class is governed by the following vague sentence: Two objects are similar if

"Most of the important attributes of the class exhibit similar values in the objects."

This vague sentence matches the template:

\[ Q \text{ of } D \text{ are } A \]

of a type II sentence of Zadeh’s calculus of linguistically quantified statements, where \( Q \) is a linguistic quantifier and \( A \) and \( D \) are vague properties, both defined as fuzzy sets over a finite reference universe \( X \) with membership functions \( \mu_A \) and \( \mu_D \), respectively:

- \( Q \) is the quantifier \textit{most}.
- \( X = \{a_1, a_2, \ldots, a_n\} \) is the set of attributes that characterize the type of the class.
- \( D \) is the set of attributes that are relevant for the compatibility computation:
  \[ \mu_D(a_i) = p_{a_i}, \forall a_i \in X \]
- \( A \) is the set of attributes that have similar values in both objects:
  \[ \mu_A(a_i) = S_{a_i}(o_1, o_2), \forall a_i \in X \]

3. Operators

In this section we look at various operators which may be potentially useful for the computation of the compatibility of objects.

We illustrate the applicability of this operators on a fictitious example related to crime investigations. Such an application is meant to help build a profile of a criminal who committed a crime under investigation (perpetrator). We assume that in this application we maintain a data structure containing profiles of suspects and compare them with a constructed profile of the perpetrator resulting from the testimonies of witnesses. A profile is represented as an object of the PROFILE class. This class possesses typical attributes like “age”, “height”, “hair-colour”, etc.

3.1. Compatibility operators

As mentioned earlier, the first step in judging the compatibility of two objects is a comparison of values of their attributes. Depending on the type of attribute values (cf. Figure 1) and a required compatibility level, various operators may be employed. In what follows we will review a repertoire of operators for particular pairs of types of attribute values.
The basic operator in this context is the equality. This is, of course, a standard comparison operator available on all software platforms, and, thus, less interesting for our considerations.

Often the equality of attribute values is not required. For example, comparing the perpetrator’s profile with one of a suspect, even having testimonies on a precise color of perpetrator’s hairs, the application may need to accept some differences between attributes values of two profiles (objects) (for example, this may be useful in case there is no exact match with any of our suspects but still we are somehow sure one of them is the actual perpetrator).

In order to provide a support for such an approximate or inexact equality the concept of compatibility may be associated with a domain of given attribute. There are various means proposed in the literature to model compatibility:

- The most natural would be to adopt here Zadeh’s original concept of a (fuzzy) similarity relation. This is defined as a reflexive, symmetrical, and transitive relation (transitivity may be interpreted in various ways).

Nevertheless, the concept of fuzzy similarity relation does not seem to be a good model of compatibility for the purposes of object comparison, mainly because of its strictness.

- Another related concept referred to in the literature is that of a resemblance relation (cf., e.g., Ref. 9). The interpretation of this concept does not seem to be universally agreed upon by different authors. Usually, it is assumed to be somehow less restrictive than the similarity relation by a relaxation of the transitivity property.

- For our purposes, it seems that the best model of compatibility is a proximity (also known as a tolerance) relation, i.e., a reflexive and symmetric relation (although, even the latter property is contested by some authors).

Practically, two cases should be distinguished, namely those with a finite and infinite (continuous) domain. In the former case the proximity relation may be enumerated, i.e., all pairs of domain elements belonging to the relation may be listed explicitly. In the latter, a formula for the degree of proximity of any two elements of a domain should be provided.

According to the above discussion, if \( o_1 \) and \( o_2 \) are the objects we are comparing, \( a_i \) is the attribute under study, and \( \text{Prox} \) is the proximity relation defined on \( a_i \) domain, then:

\[
S_{a_i}(o_1, o_2) = \mu_{\text{Prox}}(o_1.a_i, o_2.a_i)
\]

In practice, many forms of imperfect information need to be modeled for the purposes of various applications. Here, we would like to focus on the representation of
uncertainty using possibility distributions. This concept, originally introduced by Zadeh, perfectly fits into our illustrative example.

For instance, it is quite common that a witness describes a perpetrator as, e.g., a “young, medium high guy”. These linguistic terms may be appropriately modeled using some fuzzy sets defined in the domains of age and height, respectively. Effectively, they impose a possibility distribution on the values of corresponding attributes.

In this context, the compatibility of a precise, scalar value $x$ and an imprecise value of an attribute $A$ represented using a possibility distribution $\pi_A$ may be best accounted for by $\pi_A(x)$.

Thus, if $o_1$ and $o_2$ are the objects we are comparing, $a_i$ is the attribute under study, $o_1.a_i$ is a precise value, and $\pi_{o_2.a_i}$ is a possibility distribution defined on $a_i$ domain, then:

$$S_{a_i}(o_1, o_2) = \pi_{o_2.a_i}(o_1.a_i)$$

**Imprecise value, Imprecise value**

Although the terms “imprecise value” is wide and covers a large variety of cases, we will focus here on two main cases:

- First, let us assume we want to compare two attribute values represented by possibility distributions. For example, we may be in need to compare two suspects who were described by the witnesses as young and around 25 years old. Thus, effectively we need to assess the possibility (as well as necessity) that two variables $X$ and $Y$, whose values are given by possibility distributions, $\pi_X$ and $\pi_Y$, are in relation $\theta$ (here: equality, similarity, proximity, ...). In the framework of the possibility theory this may be done as follows.
  - First, the joint possibility distribution, $\pi_{XY}$, of $X$ and $Y$ on $U \times U$ (assuming non-interactivity of the variables) is determined as:
    $$\pi_{XY}(u, w) = \min(\pi_x(u), \pi_y(w))$$
    Relation $\theta$ is represented by a fuzzy set $F$ such that
    $$\mu_F(u, w) = \mu_\theta(u, w)$$
    The possibility (necessity) measure associated with $\mu_{XY}$ will be denoted by $\Pi_{XY}(N_{XY})$.
  - Then, we calculate the measures of the variables $X$ and $Y$ being in relation $\theta$ as follows:
    $$\text{Pos}(X \theta Y) = \Pi_{XY}(F) = \sup_{u, w \in U} \min(\pi_X(u), \pi_Y(w), \mu_\theta(u, w))$$
    $$\text{Nec}(X \theta Y) = N_{XY}(F) = \inf_{u, w \in U} \max(1 - \pi_X(u), 1 - \pi_Y(u), \mu_\theta(u, w))$$
A General Framework for Computing with Words in Object-Oriented Programming

This formula may be seen as the value-based comparison of attribute values (represented using possibility distributions). In some cases, the representation-based (cf., Bosc et al.\textsuperscript{3}) comparison may be applicable. This approach consists in direct comparison of the representation of the attributes values, i.e., features of the corresponding possibility distributions rather than the values themselves. For example, we may find two perpetrator profiles equally promising if they have the same number of completely possible (to a degree 1) values for given attribute.

In summary, for example, if we take $\text{Pos}(XY)$ as our desired comparison value, and $(o_1, o_2)$ is the couple of objects we are comparing, $a_i$ is the attribute under study, $o_1.a_i$ and $o_2.a_i$ are imprecise values expressed in terms of possibility distributions defined on $a_i$ domain (namely, U), then:

$$S_{a_i}(o_1, o_2) = \text{Pos}(o_1.a_i; o_2.a_i) = \Pi_{o_1.a_i, o_2.a_i}(F) = \sup_{u, w \in U} \min(\pi_{o_1.a_i}(u), \pi_{o_2.a_i}(w), \mu(u, w))$$

- Second type of comparison of two imprecise values refers to the case when values of attributes may be fuzzy sets (i.e., basically, multiple-valued attributes). For example, an attribute in a profile may represent languages spoken by given person. Then, using a fuzzy set as the value of such an attribute, particular membership function values of various languages may correspond to the level of the command of these languages by the person represented with a given object (profile).

  Cross and Sudkamp’s\textsuperscript{4} book provides a comprehensive coverage of various approaches to the measuring of similarity and compatibility of fuzzy sets, and – for our purposes – one can distinguish here the following main classes of measures which result from some milder conditions. Basically, we may have: inclusion indices, partial matching indices, similarity indices, and proximity-based measures.

  Less limiting are inclusion indices. First, one introduces a so-called scalar evaluator $g$ which reduces a fuzzy set, say $A$ in $X$ to a real number from $[0, 1]$. The following are natural conditions to be satisfied by $g$ (cf. Dubois and Prade\textsuperscript{6}):

  * $g(\emptyset) = 0$,
  * $g(X) = 1$, and
  * if $A \subset B$, then $g(A) \leq g(B)$.

  Then, an inclusion index between $A$ and $B$ in $X$ is supposed to satisfy:

  * $I(A, B) = 1$ if and only if $\overline{A} \cup B = X$,
  * if the supports of $A$ and $B$ are disjoint, then $I(A, B) = 0$, and
  * $I(A, B)$ depends on a scalar evaluation of $\overline{A} \cup B$, $g(\overline{A} \cup B)$.

  A general formula for a fuzzy set inclusion index, that will satisfy the above conditions, would be:

  $$I(A, B) = \frac{g(\overline{A} \cup B) - g(\overline{A})}{1 - g(A)}$$ (1)
Notice that a general formula for a particular class of indices (measures) is of much relevance for the purpose of this work as we try to develop a general framework that might help implement a general paradigm of computing with words, and the flexibility we obtain in such a way is of much practical importance.

In our comparison context, if \( o_1 \) and \( o_2 \) are the objects we are comparing, \( a_i \) is the attribute under study, \( o_1.a_i \) and \( o_2.a_i \) are the fuzzy sets defined on \( a_i \) domain, then:

\[
S_{a_i}(o_1, o_2) = I(o_1.a_i, o_2.a_i) = \frac{g(o_1.a_i \cup o_2.a_i) - g(o_1.a_i)}{1 - g(o_1.a_i)}
\]

The partial matching indices do not require a high degree of similarity. They are a direct generalization of the traditional Zadeh’s equality of two fuzzy sets \( A \) and \( B \) in \( X \) which are said to be equal when \( \mu_A(x) = \mu_B(x) \), for all \( x \in X \). Then, since this measure of equality is very strict, it can be weakened, for instance to the classic Zadeh’s compatibility index (with values in \([0,1]\)):

\[
\text{comp}(A,B) = \sup_{x \in X} \min(\mu_A(x), \mu_B(x))
\]

In general, the conditions imposed on a partial matching index between two fuzzy sets \( A \) and \( B \) in \( X \), \( P(A, B) \), are usually as follows:

* \( P(A, B) = 0 \) if and only if \( A \cap B = \emptyset \),
* \( P(A, B) = 1 \) if \( A \subseteq B_{\alpha=1} \) or \( B \subseteq A_{\alpha=1} \),
* \( P(A, B) = P(B, A) \),
* \( P(A, B) \) depends only on \( g(A \cap B) \), where \( g \) is a scalar evaluator.

The general formulas for a partial matching index that satisfies the above conditions is:

\[
P_{\cap/g}(A, B) = \frac{g(A \cap B)}{\min(g(A), g(B))}
\]

According to our notation, if \( o_1 \) and \( o_2 \) are the objects we are comparing, \( a_i \) is the attribute under study, \( o_1.a_i \) and \( o_2.a_i \) are the fuzzy sets defined on \( a_i \) domain, then:

\[
S_{a_i}(o_1, o_2) = P_{\cap/g}(o_1.a_i, o_2.a_i) = \frac{g(o_1.a_i \cap o_2.a_i)}{\min(g(o_1.a_i), g(o_2.a_i))}
\]

A similarity index requires a greater degree of agreement between two fuzzy sets. Usually, the following conditions are imposed (cf. Dubois and Prade\(^6\)) on a similarity index between two fuzzy sets \( A \) and \( B \) in \( X \), \( S(A, B) \):

* \( S(A, B) = 1 \) if and only if \( (A \cup B) \cap (\overline{A} \cup \overline{B}) = \emptyset \),
* if \( A \) and \( B \) have disjoint supports, then \( S(A, B) = 0 \),
* \( S(A, B) = S(B, A) \), and
* \( S(A, B) \) depends on \( g(A \cup B) \cap (\overline{A} \cup \overline{B}) \) or on a symmetric function of \( g(A \cup B) \) and \( g(\overline{A} \cup B) \).
A General Framework for Computing with Words in Object-Oriented Programming

The general formulas for a similarity index can be different (cf. Cross and Sudkamp\(^4\)), and a widely used one is:

\[
S(A, B) = \frac{g((A \cup B) \cap (\overline{A} \cup \overline{B}) - g(A \cup B))}{1 - g(A \cup B)}
\]

(3)

One of more widely used indices of similarity is here the Jaccard index:

\[
S_J(A, B) = \frac{||A \cap B||}{||A \cup B||}
\]

(4)

where \(|| \cdot ||\) denotes the cardinality of the respective fuzzy set.

We can apply this result to our comparison problem. Once again, if \(o_1\) and \(o_2\) are the objects we are comparing, \(a_i\) is the attribute under study, \(o_1.a_i\) and \(o_2.a_i\) are the fuzzy sets defined on \(a_i\) domain, then:

\[
S_{a_i}(o_1, o_2) = S_J(o_1.a_i, o_2.a_i) = \frac{||o_1.a_i \cap o_2.a_i||}{||o_1.a_i \cup o_2.a_i||}
\]

Finally, one can define measures that are based on proximity (cf. Cross and Sudkamp\(^4\)). Basically, these compatibility measures need not satisfy the conditions of a metric. We will not consider them here for lack of space.

(Fuzzy Collection, Fuzzy Collection)

We have seen how to deal with conjunctive semantics in imprecise attribute values defined on a precise domain \(X\). However, in an object-oriented context, it may happens that the elements of \(X\) are fuzzily described objects. That is, we have fuzzy collections of fuzzily described objects. In these situations, for a given element in the set \(A\) it is not clear which element of \(B\) has to be taken in order to compare the membership degrees.

In our example of PROFILES, we can have the description of a gang together with the description of a given criminal. The gang could be a fuzzy collection of criminals (described by their fuzzy profile). In order to compare two criminals, we have to compare their gangs. In this last comparison, two different criminal objects (from the identity equality point of view) may be the same one (from the value equality point of view), and we have to take this fact into account when comparing both collections of criminals.

A suitable approach to compare fuzzy collections of fuzzily described objects can be found in Ref. 10. We can face the conjunctive fuzzy set comparison by means of the concept of inclusion:

\[
A = B \text{ if, and only if, } (A \subseteq B) \land (B \subseteq A)
\]

(5)

To compute the inclusion degree of a fuzzy set \(A\) in a fuzzy set \(B\) we can use the following operator\(^13\):

\[
N(B|A) = \min_{x \in X} \{I(\mu_A(x), \mu_B(x))\}
\]

(6)
where $I$ stands for a fuzzy implication operator and $X$ is the reference set where $A$ and $B$ are defined.

If $X$ is a set of fuzzily described objects, in order to perform comparison among fuzzy collections of fuzzily described objects, the following set of operators is proposed in Ref. 10:

- An inclusion operator ($\Theta$), which takes into account resemblance between the elements which are being compared:
  \[
  \Theta_S(B|A) = \min_{x \in X} \max_{y \in X} \theta_{A,B,S}(x, y)
  \]
  where
  \[
  \theta_{A,B,S}(x, y) = \Theta(I(\mu_A(x), \mu_B(y)), \mu_S(x, y))
  \]

- A generalized resemblance operator ($\exists$), which considers both inclusion directions and which can be weighted with a cardinality ratio ($\Phi$):
  \[
  \exists_{S,\Theta}(A, B) = \Theta(\Theta_S(B|A), \Theta_S(A|B))
  \]
  \[
  \Phi(A, B) = \begin{cases} 
  1, & \text{if } A = \emptyset \land B = \emptyset \\
  \frac{\min(|A|, |B|)}{\max(|A|, |B|)}, & \text{otherwise}
  \end{cases}
  \]

We can apply this result to our comparison problem. Once again, if $o_1$ and $o_2$ are the objects we are comparing, $a_i$ is the attribute under study, $o_1.a_i$ and $o_2.a_i$ are the fuzzy collections of fuzzily described objects, and $S$ is the relation that provides resemblance values among the elements of the collections, then:

\[
S_{a_i}(o_1, o_2) = \exists_{S,\Theta}(o_1.a_i, o_2.a_i)
\]

As can be seen, the presented operator requires a high degree of similarity between the compared collections. However, its behavior can be configured according to the used Implication, T-norm, and T-conorm.

### 3.2. Aggregation operators

The application of methods discussed in the previous subsection yields a compatibility degree for each relevant attribute characterizing objects under comparison. In order to arrive to a global compatibility index we need to aggregate thus obtained compatibility degrees. As mentioned in Section 2.3, our preferred aggregation operators are linguistic quantifiers. However, there are many other options, i.e., various aggregation operators elaborated on the ground of fuzzy logic context and not only there. Here, we will briefly discuss some of them.

The simplest aggregation operators are the AND and OR logical connectives. In the fuzzy logic they are often represented by $\min$ and $\max$ operators, respectively, while in general, any t-norm and t-conorm, respectively, may be used. However, for many applications such an aggregation mode may be inadequate being too restrictive or too tolerant.
Salton et al.\textsuperscript{14} proposed to make the AND operator more flexible, stating that the total compatibility of objects is equal to the complement to 1 of the normalized distance between points \( (S_a(o_1, o_2), S_{a_2}(o_1, o_2), \ldots, S_{a_n}(o_1, o_2)) \) and \((1, 1, \ldots, 1)\) in a \( n \)-dimensional metric space, where \( n \) is the number of relevant attributes. A more flexible version of the OR operator may be analogously obtained considering \((0, 0, \ldots, 0)\) instead of \((1, 1, \ldots, 1)\). Obviously, different variants of distance definition lead to different flexible operators.

As mentioned earlier the concept of a flexible majority expressed with the use of a linguistic quantifier is our aggregation operator of choice. However, it may be modelled in various ways, including a well-known calculus of linguistically quantified propositions of Zadeh\textsuperscript{20} or Yager’s OWA operators.\textsuperscript{15} There are many more approaches proposed in the literature. In the ordered weighted min (OWMin)\textsuperscript{7} operator the concept of a required majority is modeled as a fuzzy set \( I \) in the space \( \{0, 1, 2, \ldots, n\} \) (\( n \) is the number of relevant attributes), such that \( \mu_I(0) = 1; \mu_I(i) \geq \mu_I(i+1) \). Thus, if for a global compatibility of two objects \( o_1 \) and \( o_2 \) it is, for example, required that “the values of at least \( k \) attributes are similar”, then one sets \( \mu_I(i) = 1 \) for all \( 0 \leq i \leq k \) and \( \mu_I(i) = 0 \) for all \( i > k \). The values to be aggregated, i.e., \( S_{a_i}(o_1, o_2) \)’s, are sorted in a non-increasing order yielding a vector \((s_1, s_2, \ldots, s_n)\) such that \( s_j \) is the \( j \)-th highest from among \( S_{a_i}(o_1, o_2) \)’s. Then, the global compatibility of the objects \( o_1 \) and \( o_2 \) is given by: \( \min_{i=1,n} \max(1-\mu_I(i), s_i) \).

Now, let us briefly discuss how the relevance (importance) degree of the attributes may be taken into account by aggregation operators. Let us start again with classical aggregation operators AND and OR generalized with \( t \)-norm and \( t \)-conorm operators. We will focus on the former – for the latter an analogous discussion may be carried out.

The idea is still to use the min (or another \( t \)-norm) operator, but for slightly modified arguments. Fairly general formula for the incorporation of the attributes relevance degrees, \( p_{a_i} \), is obtained by replacing \( S_{a_i}(o_1, o_2) \) with \( p_{a_i} \rightarrow S_{a_i}(o_1, o_2) \), cf., Ref. 7. Various implication operators, \( \rightarrow \), may be assumed in this formula, depending on the context in which the objects are compared. For the Kleene-Dienes implication \( (x \rightarrow y = \max(1 - x, y)) \) we obtain a so-called relative importance interpretation. Namely, if an attribute, \( a_i \), is completely unimportant \( (p_{a_i} = 0) \), then its values for compared objects are irrelevant for objects’ compatibility. On the other hand, if an attribute is very important \( (p_{a_i} = 1) \), then the values of this attribute for both objects have to be very similar \( (S_{a_i}(o_1, o_2) \) close to 1) in order to secure full total compatibility of objects \( o_1 \) and \( o_2 \). Summarizing, the more important attribute the higher its influence on the compatibility of the objects.

For the Gödel implication \( (x \rightarrow x = 1 \text{ if } x \leq y, \text{ otherwise}) \) we get the threshold interpretation of the importance weights. Namely, to preserve a full total compatibility of two objects their compatibility degree at each attribute have to be not lesser than this attribute’s importance weight.

The use of still another implication operator, the Goguen implication \( (x \rightarrow y \)
= 1 if $x \leq y$, $y/x$ otherwise) supports the same “threshold logic” with a different quantification of the situation where the compatibility degree at a given attribute does not reach this attribute’s importance weight.

A similar analysis may be provided for the OR-type aggregation using the coimplication instead of the implication operator.

The question of linguistic quantifier guided aggregation with importance weights has been deeply studied in the framework of both Zadeh’s calculus of linguistically quantified propositions and Yager’s OWA operators. In the former case, type II propositions (sentences) are considered. In the latter case, the problem is more complex.

4. Dealing with Cycles

In order to deal with the resemblance calculus between complex objects, the only solution is to propagate the problem by means of recursion. Suppose that we are comparing objects $o_1$ and $o_2$, and the values of a given attribute in these objects are objects $o_3$ and $o_4$. To compute the resemblance between $o_1$ and $o_2$, we previously need to compute the resemblance between $o_3$ and $o_4$, and so on.

The use of recursion is not a problem unless there are cycles in the relationships graphs. For instance, let us consider the graph in Figure 3. Suppose that we have two objects of class A, namely, $o_{A1}$ and $o_{A2}$, and we want to calculate their resemblance degree. When we analyze attribute $a$, we will need to study the resemblance degree between two objects of the class B, for example $o_{B1}$ and $o_{B2}$. When trying to solve this latter resemblance, we will need to study the resemblance between two objects $o_{C1}$ and $o_{C2}$. Finally, to know the resemblance between these objects, we will need to compute the resemblance between two objects of the initial class A.

![Diagram](image)

Fig. 3. The problem of cycles.

The cycle in the above example introduces the following problems:

- Firstly, we may have to solve a comparison problem in the same class that was our starting point. This increases the complexity of the problem and suggests the necessity of exploring a wide data set in order to establish the objects resemblance.
Secondly, it may be that the resemblance degree of the objects we want to compare is part of the recursive tree generated in the computation. For example, if attribute c led back to object $o_{A1}$ and $o_{A2}$ again, we would have an infinite cycle.

The first problem is unavoidable because is due to the high interrelation of data in the object-oriented model. The second is far more dangerous because it prevents us from ending the computation. We can solve it by means of the following:

- Not propagating recursively, ignoring this in the general calculus.
- Approximating the value in some way, making several iterations until reaching the final value.
- Directly approximating the value with another semantically valid one.

The first alternative (not propagating recursively) is not suitable because we may be ignoring important information. Although the second alternative (the use of an initial approximation and then iterate) is acceptable it requires a considerable calculus effort and a higher algorithmic complexity (more than one cycle may appear in a normal recursion process).

Our proposal is to use the third alternative. We can unfold the objects resemblance in two different ways: one that expresses the surface resemblance between the objects and the other based on an object exploration in depth. The first will be based on the object attributes which will not involve the cycling problem and the other will be a resemblance obtained taking into account the attributes that need recursive monitoring. The surface resemblance can be used as an approximation when, in the calculus of the second one (the real resemblance), a cycle is detected.

5. Programming with Words

The implementation of the aforementioned comparison process is complex and, as we have seen, it needs a recursive computation which might involve cycle resolution (e.g. for comparing criminal PROFILES, we may have to compare gangs or even victim descriptions).

We provide programmers with a framework which enables the application of the ideas described above in real applications. Our fuzzy objects library simplifies the creation of special domains to represent fuzzy attribute values, as well as the implementation of user-defined classes which use those values. It also provides the infrastructure which allows the comparison of fuzzily-described objects by making use of well-known object-oriented design principles.

5.1. An extensible framework

Our framework incorporates a predefined class hierarchy that supports the representation of fuzzily-described objects. The root in this hierarchy is a generic FuzzyObject class which serves as a basis for any class requiring fuzzy comparison capabilities. This class implements a generic FuzzyEquals method which performs
the fuzzy object comparison described in the previous sections. This method can be used to compare objects belonging to any subclass of FuzzyObject. Since the comparison it performs requires access to the particular fields of the objects being compared and, in order to be reusable, it must not be fitted to any particular class structure, the method implementation uses reflection to perform the object comparison. The comparison algorithm is recursive in order to manage complex objects and is designed so that it seamlessly deals with the cyclic structures which are common in object graphs.

Below the FuzzyObject class in our framework class hierarchy, shown in Figure 4, other predefined classes represent common kinds of domains for handling imprecision, such as linguistic labels without an underlying representation (DomainWithoutRepresentation), domains where labels are possibility distributions over an underlying basic domain (DisjunctiveDomain), and fuzzy collections of fuzzy objects (ConjunctiveDomain). All these classes define their own FuzzyEquals method, whose implementation uses suitable operators depending on the semantics of the domains they represent according to the discussion of Section 3.

5.2. Customizing the framework

Current programming platforms provide means for adding declarative information (metadata) to runtime entities such as classes, methods, and instance or class variables. Metadata is stored with your program at compile time and can be retrieved and used at runtime (through Reflection). Both Microsoft’s .NET Framework and Sun Microsystems’s Java Platform support the definition and use of metadata by means of custom attributes and annotations, respectively.

In the .NET Framework, programmers can create their own custom attributes by creating subclasses of the System.Attribute class. For instance, in order to support the adaptable use of aggregators in FuzzyEquals, we created two attributes in $C^*_2$:

- The Aggregator operator can be used by the programmer to set the way compatibility degrees of attribute values should be aggregated. The user will provide via this attribute the name of a class which implements the desired operator (this class must implement the interface Aggregator included in the framework, which is a generalization of a function $R^n \rightarrow R$).
The *Weighting scheme* operator can be used by the programmer to set the way importance degrees of attributes have to be taken into account in the comparison process. The user will provide via this attribute the name of a class which computes the desired behavior (this class must implement the interface $\text{Wscheme}$ included in the framework, which is a generalization of a function $R^n \times R^n \rightarrow R^n$).

The previous attributes can be applied to user-defined classes. Our framework also defines another metadata attribute, *FuzzyImportance*, which can be used in accessor methods to indicate the weight of a property in the comparison process.

Similar attributes are defined to generalize the use of $\text{DomainWithoutRepresentation}$, $\text{DisjunctiveDomain}$, and $\text{ConjunctiveDomain}$.

Figure 5 shows the operation of the framework. A dynamic linking library that enables the use of fuzzily described objects in the code has to be added to the project the programmers create to develop the desired soft computing application. An extensible repository of built-in operators complete the development facilities and permits to adapt the behavior of the comparison framework to the particular necessities of the soft computing application under development. The user only have to write *standard* C$\sharp$ code in order to develop the desired soft computing functionalities, being able to friendly manage imprecise data without having to worry about their underlying support.

**A simple example**

A soft computing application dealing with criminal profiles can be easily developed in C$\sharp$ using our framework. $\text{Profile}$, $\text{Gang}$, and $\text{Victim}$ imprecise attributes can be easily implemented just by extending the classes provided by our framework without having to worry about the $\text{FuzzyEquals}$ implementation.

For instance, $\text{Age}$ and $\text{Height}$ domains for a criminal $\text{Profile}$ can be easily implemented just by extending the predefined $\text{DisjunctiveDomain}$ class.

Once the attribute domains are defined, the $\text{Profile}$ class can be easily implemented using $\text{FuzzyObject}$ as its base class, as shown in the following code. As you can see, the programmer does not have to write any code for object comparison. That feature is automatically provided by the $\text{FuzzyObject}$ base class.

```csharp
[Aggregator(typeof(OWA), typeof(Most))]
[Weighting Scheme(typeof(Min))]
public class Profile : FuzzyObject {
    // Instance variables
    private Age age;
    private Height height;
    ...
    // Constructor
    public Profile (Age age, Height height, ...) {
        this.age = age;
    }
```
public class Profile : FuzzyObject
{
    // Instance variables
    private Age age;
    private Height height;
    ...
    // Constructor
    public Profile(Age age, Height height, ...) {
        this.age = age;
        this.height = height;
    }
    // Properties
    [FuzzyImportance(0.8f)]
    public Age ProfileAge {
        get { return age; }
        set { age = value; }
    }
    [FuzzyImportance(0.7f)]
    public Height ProfileHeight {
        get { return height; }
        set { height = value; }
    }
    ...
}

Fig. 5. The development of soft computing applications on standard programming platforms.

this.height = height;
...

// Properties
[FuzzyImportance(0.8f)]
public Age ProfileAge {
    get { return age; }
    set { age = value; }
}
[FuzzyImportance(0.7f)]
public Height ProfileHeight {
    get { return height; }
    set { height = value; }
}
...

Using the attributes **Aggregator** and **Weighting Scheme**, we indicate that we want to use classes **OWA** and **Min** to perform the aggregation in the comparison process. This classes must be previously defined in the system: they can be taken from the operator repository or can be directly coded by the programmer according to the interfaces we previously mentioned. The attribute **FuzzyImportance** is used to set the importance of Age and Height to 0.8 and 0.7, respectively.

Once the class **Profile** has been created, objects representing criminals can be created using standard C# code:

```csharp
Age young = new Age ( new Label("young"), 0, 0, 23, 33 );
Height tall = new Height ( new Label("tall"), 170, 180, 300, 300);
...
Profile c1 = new Profile (young, new Height(185), ...);
Profile c2 = new Profile (middle, tall, ...);
```

Our framework reflective implementation automatically enables object comparison by means of the **FuzzyEquals** method. Thus, in order to compare the two profiles, the user just needs to invoke this method:

```csharp
c1.FuzzyEquals(c2).
```

6. Conclusions

This paper has studied the problem of fuzzily-described object comparison. We have provided both a general approach to perform this comparison as well as a comprehensive survey of operators from the literature that can be used to customize the comparison process.

The paper has also presented a general framework that allows the implementation of soft computing applications dealing with fuzzily-described objects. In order to maximize our framework user-friendliness, we have taken advantage of some advanced capabilities provided by modern programming platforms such as Microsoft .NET Framework (namely, reflection and metadata attributes).

Our framework allows for the use of different operators in the comparison logic, let them be predefined or user-defined.

Our future research is now focused on the completion of the operators repository, so that our framework users can make use of a comprehensive variety of operators according to the needs of the soft computing application they must develop without having to code them. We are also extending our general framework in order to allow the use of linguistic labels in the source code description of fuzzily-described objects (e.g. in the definition of fuzzy importance values for class attributes).

Acknowledgements

This paper has been supported in part by the Spanish “Comisión Interministerial de Ciencia y Tecnología” under grants TIC2003-08687-C02-02 and TIC2002-04021-C02-02.
This paper has also been supported in part by the Polish State Committee for Scientific Research grant 3 T11C 052 27.

References