Managing Fuzziness on Conventional Object-Oriented Platforms

F. Berzal,[†] N. Marín,[*] O. Pons,[‡] M.A. Vila[§]
IDBIS Research Group, Department of Computer Science and A.I., E.T.S.I.I., University of Granada, 18071, Granada, Spain

During the last few years, many database researchers have aimed their efforts at extending the object-oriented model for dealing with different kinds of imperfect information. Some of these scholars have used the Fuzzy Set Theory to deal with imperfection because it has proved to be useful in problems where imprecision and uncertainty play important roles. This article describes an architecture that can be used to develop a fuzzy object-oriented system on top of an existing classical one. This article also introduces a general framework as the basis for managing fuzziness in conventional object-oriented systems. Foodbi, a fuzzy object-oriented database interface, is presented as a prototype that allows the creation of fuzzy object-oriented schemata that can be translated into sets of standard Java classes. © 2007 Wiley Periodicals, Inc.

1. INTRODUCTION

Real-world data not only present complex structures, but they are usually affected by imperfections of very different kinds. The Fuzzy Set Theory[1] has proved to be an adequate tool in handling these situations. The application of this theory in order to extend conventional databases has led to the development of fuzzy database systems.

Object-oriented and object-relational database management systems allow the representation of schemata when complex relationships make the use of Codd's relational model difficult. The object-oriented data model is more powerful from a modeling point of view because it incorporates important features such as inheritance and encapsulation. As was the case with the relational data model in the past, many researchers have recently tried to improve object orientation with the help of fuzzy concepts. As a result, fuzzy object-oriented database models (FOODBM) have appeared.[2,3]

*Author to whom all correspondence should be addressed: e-mail: nicm@decsai.ugr.es.
[†]e-mail: fberzal@decsai.ugr.es.
[‡]e-mail: opc@decsai.ugr.es.
[§]e-mail: vila@decsai.ugr.es.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 22, 781–803 (2007)
 Published online in Wiley InterScience (www.interscience.wiley.com). • DOI 10.1002/int.20228

1.1. Imprecise and Uncertain Data: A Common Type of Imperfection

There is a wide variety of imperfections that can affect data. In this work, we focus on the types of imperfections that lead to data with a "soft" appearance, far from a precise and completely certain specification. Different words have been used to deal with these matters:

- The term *imprecision* denotes the lack of exactness in the expression of the information. For example, "The image is old" is more imprecise than "The car is five meters long."
- The term *uncertainty* denotes the situation that arises when we are not sure about the veracity of the information. For example, "It is possible that the car is five meters long" provides information affected by uncertainty.
- Some authors use the term *vagueness* as a synonym for imprecision. However, other authors consider that information is vague when it is affected by imprecision, uncertainty, or both of them.

The bounds between imprecision and uncertainty are difficult to establish. We can say that both characteristics are convertible.[4] If we have imprecise information, as in "The car is big," we do not actually know the size of the car with certainty, that is, the imprecise information has implicit uncertainty. A more precise value can be used to express the size of the car, but there is still some explicit uncertainty. That is the case of the sentence "It is very possible that the car is five meters long."

Many forms of data imperfection involve the notion of gradualness. In many situations, the concepts we use to describe the situation do not correspond to a simple crisp reality, but a gradual one. For example, concepts such as *youth* are gradual: from a given population, we cannot say what people are young and what people are not. However, given a particular person, we can say if he is more or less young. The concept of *youth* is gradual. Some people are undoubtedly young, some other people are undoubtedly not young, and many people are in between. A soft bound exists.

It is important to remark that imperfect data include more types of data deficiencies than the aforementioned ones. For example, erroneous or inconsistent data are also important types of imperfection that can appear in databases. These other kinds of imperfection deserve a different approach and they are, therefore, out of the scope of our model.

1.2. Fuzzy Information in the Object-Oriented Model

The concept of *object* is the core element of the object-oriented model. An object is the result of the encapsulation of the state (structure) and the conduct (behavior) of a given real-world item. Structure and behavior of similar objects are represented and reused by means of the definition of classes. Classes are organized in inheritance hierarchies, where structural and behavioral definitions are inherited among classes. Inheritance allows the definition of superclasses and subclasses.

Fuzzy Set Theory has been used in the context of object-oriented databases in order to improve their modeling capabilities, so that different kinds of data imperfections can be represented. Fuzzy object-oriented database models have appeared where fuzziness has been studied at different levels[5]: domains, instance relationships, inheritance relationships, structure, and behavior.

Advanced semantic data models can be considered as the origin of the study of fuzziness in object-oriented models (e.g., the Fuzzy Entity/Relationship Model[6–8]). Rossazza et al.[9] introduced a hierarchical model of fuzzy classes, explaining important notions by means of the use of fuzzy sets (e.g., the typicality concept). George et al.[10] began to use similarity relationships in order to model attribute value imperfection. The work of George et al. has been completed by Koyuncu and Yazici,[11] with IFOOD as a result, an intelligent fuzzy object-oriented data model.

In the past decade, Fuzzy Object-Oriented Database Models became an independent research field within the database research area: Bordogna et al.[12] introduced an extended graphical notation to represent fuzzy object-oriented information, whereas Van Gyseghem and De Caluwe[5] developed the UFO model, one of the most complete proposals that can be found in the literature.

1.3. Target and Organization

The fuzzy object-oriented data models cited in the previous paragraphs are of remarkable value for two main reasons: their experimental development has been the basis for the study of the appropriateness of the use of fuzzy concepts in an object-oriented context, and they are powerful modeling tools for those interested in working with this kind of advanced database model.

However, if the user wants to use the fuzzy structures proposed by those models, he needs the corresponding DBMS. To ease this development task, we propose the use of a conventional object-oriented database model as the platform for building fuzzy object-oriented schemata with the common data imperfections we have mentioned before.

This article introduces an architecture that can be used to develop a system able to store fuzzy information in conventional object-oriented systems. The main component of this architecture is a framework developed using conventional object-oriented capabilities. This framework supports the fuzzy extension of many classical object-oriented features. It can be transparently used through an interface in order to provide fuzzy object-oriented data management capabilities to users. To complete the description of our approach, we present Foodbi, a fuzzy object-oriented database interface that allows the creation of fuzzy object-oriented schemata that can later be translated into sets of standard Java classes.

Our article is organized as follows. Section 2 shows the general strategy we follow in our development and Section 3 explains the different extensions to classical object-orientation we have considered in our system. Section 4 is devoted to the prototype interface Foodbi. Finally, some conclusions and future work suggestions are presented in Section 5.

2. FUZZINESS ON CONVENTIONAL DATABASE SYSTEMS

In Ref. 13, we presented a fuzzy object-oriented data model that handles fuzziness at most levels where it can appear in an object-oriented context. The model does not only allow the representation of different kinds of fuzzy domains in the

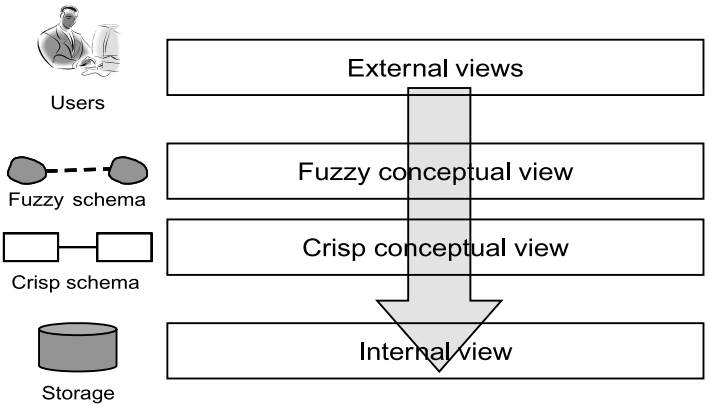

Figure 1. Fuzzy views on top of a conventional database system.

database but it also introduces some fuzzy extensions of classical object-oriented concepts.

The model is built following a basic principle: the use of conventional object-oriented features to implement the new fuzzy extensions. That is, in our study, all the proposed extensions of the object-oriented data model are built by means of structures that can be directly translated into a set of standard classes.

This way, we can develop a fuzzy database system using an existing conventional one as the underlying framework, with the only requirement being that it must support object-orientation.

Figure 1, based on a simplification of the ANSI/SPARC standard database architecture, depicts our strategy:

- External views are organized in such a way that the user can transparently manage data. This is the *fuzzy view* of the system.
- The conceptual schema is divided into two different layers: the higher one contains fuzzy schemata definitions, whereas the lower one holds the corresponding conventional object-oriented representation needed to support the fuzzy schemata.
- The internal schema is that of the classical database system that is being used as the basis for the fuzzy database system.

A database system supporting this layered architecture must be organized around three main components:

- A database management system, which will provide most of the management features and will store the objects created using the user-defined schemata. In our system, the DBMS must be able to deal with conventional object-orientation.
- A special API, the conceptual fuzziness handler, which will augment the system capabilities to enable fuzzy data manipulation.
- An interface, which communicates with the underlying system through the API. This interface will hide the system complexity and it will allow users to develop their fuzzy object-oriented schemata.

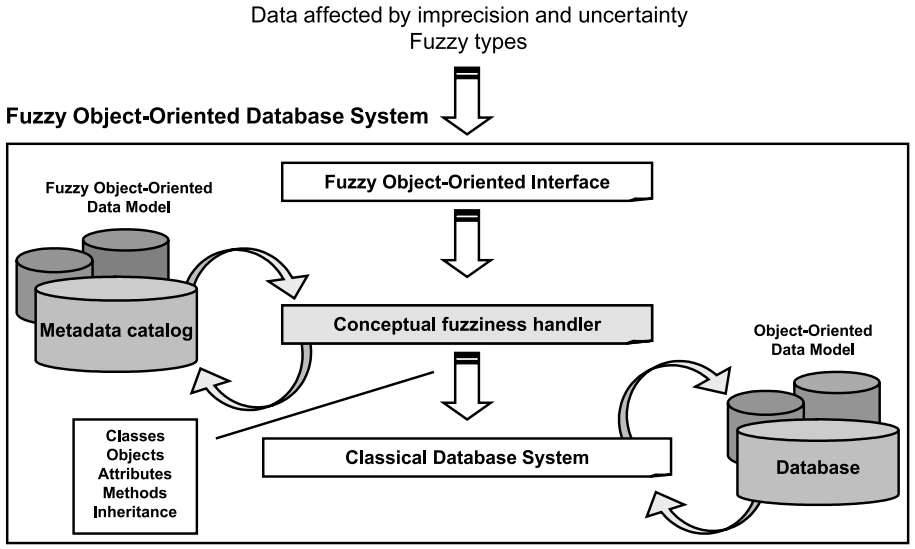

Figure 2. Fuzziness on top of a conventional database management system.

Figure 2 shows our architecture. As can be observed in this figure, metadata and general data persistence depend on two storage areas:

- a metadata catalog, which will store the fuzzy schemata defined by the user.
- a conventional database, which will support the storage and retrieval of user application objects.

The Conceptual Fuzziness Handler plays the central role in our architecture. This component is in charge of establishing the correspondence between (a) the fuzzy data and schemata, and (b) the set of standard classes and objects needed to support them.

2.1. The Fuzzy Object-Oriented Database Interface: Foodbi, a Prototype

To experiment with the architecture described above, a prototype interface has been developed. Foodbi, Fuzzy Object-Oriented Database Interface, is a graphical user interface that allows the creation and management of fuzzy object-oriented schemata. By means of this interface, the user can build a hierarchy of classes with fuzzy types and she can use, at the same time, suitable attribute domains for handling some common types of data imperfection.

Our prototype uses Java[14] as the target object-oriented language and Oracle™ as the DBMS back end.

3. THE FUZZY EXTERNAL VIEW OF THE SYSTEM

Fuzzy object-orientation provides many possibilities to handle fuzziness in a soft computing application. In our model, the fuzzy view can be built incorporating fuzziness at different levels.

3.1. Precision Levels in the Class Definition: Fuzzy Types

One of the most important aims of a designer is to find the best hierarchy of classes that represents the problem that is being modeled. A good knowledge of the problem could lead us to manage different levels of precision in the structures. This capability is offered by Foodbi: the users can design their schemata by means of fuzzy types.

3.1.1. Fuzzy Types

The structure associated to a given class can be viewed as a set of attributes or properties, with a series of associated domains. This concept of structure, called *crisp structure*, fulfills a large proportion of the needs related to types that may arise when the hierarchical structure of a given application is being developed. However, there are other problems where the use of this conventional concept of structure complicates the modeling, and a softening process could be of help. Examples of these problems are the representation of concepts with different levels of precision, semistructured or unstructured data management, or the handling of incomplete information. Designers would appreciate the use of more expressive and powerful techniques to define the structure of a certain class of objects in this kind of problems.

In Ref. 15, we presented a new concept of type (fuzzy type) that comes to improve the modeling of some of these problems, decreasing the complexity of the solution schema. Fuzzy types have been introduced as a new concept of type that allows the manipulation of concepts with different levels of precision. In a fuzzy type, properties belong to the type with a certain membership degree. This membership degree allow us to organize the properties of the type in suitable α-cuts, each of which describes the type with a different level of precision.

Let us now look at a brief summary of this concept and its most important characteristics. A wider presentation of the concept and its application to the aforementioned problems can be found in Refs. 15 and 16. Membership degrees to relate attributes with the class definition have also been used in Ref. 5.

3.1.2. The Concept of Fuzzy Type

The concept of *fuzzy type* is founded on the idea of fuzzy data structure.

DEFINITION 1. *A fuzzy structure is a fuzzy set defined over the set of all the possible attributes.*

DEFINITION 2. *A fuzzy type is a type whose structural part S is a fuzzy structure.*

Let T be a type associated to a given class C. The membership function that characterizes the structural component of the type has the following form:

$$\mu_S : Attributes -> [0,1] \tag{1}$$

where *Attributes* is the set of all attributes that can be used in our model.

The support set of any user-defined structure is always finite (it is meaningless to consider structures with an infinite support set, i.e., constituted by an infinite number of properties), so the latter function can be expressed by means of the following simplified notation:

$$S = \mu_S(a_1)/a_1 + \mu_S(a_2)/a_2, \ldots, +\mu_S(a_n)/a_n \tag{2}$$

where a_i stands for an attribute of the structure.

The complete set of attributes that might be used to characterize an instance of the type is the support set of the fuzzy set associated to the type. The kernel-set contains the basic attributes of the type, which every instance of the type must incorporate. Finally, each one of the α-cuts contains a set of attributes with which instances of the type can be created. The lower the α the higher the number of attributes of the type that are used for the instance. In this way, the type is structured in different precision levels (each one of the α-cuts).

So far, in the object-oriented model, every instance of a class could reference any of the attributes of the class (instance variables). However, with our new kind of types, an instance of a given class may not incorporate certain attributes depending on the α-cut of the class structure with which it has been created. That is, an instance of the fuzzy type cannot apply a method that uses attributes that do not belong to its structure. Therefore, we have to associate to each method of the class a degree that indicates the minimum precision (α-cut of attributes) that an instance must have to incorporate such a method in its behavior. There are two possibilities for obtaining the degree of a given method:

- The degree depends on attributes and other methods referenced in the code of the method. Thus, we can compute the degree analyzing the code of the method.

DEFINITION 3. *The precision level of the code (NC_m) of a method m is defined as the minimum membership degree of the attributes that directly appear in the code of the method, and can be calculated as follows:*

$$NC_m = \begin{cases} 1, & \text{if } Attr(m) = \emptyset \\ \min\limits_{a \in Attr(m)} \mu_S(a), & \text{otherwise} \end{cases} \tag{3}$$

where $Attr(m)$ is the set of attributes referenced in the code of the method m.

The precision level (N_m) of a method m, that is, the degree it must have associated, can be calculated by means of the following formulation:

$$N_m = \begin{cases} NC_m, & \text{if } Ref(m) = \emptyset \\ \min\left(\min\limits_{x \in Ref(m)} \{N_x\}, NC_m \right), & \text{otherwise} \end{cases} \tag{4}$$

where $Ref(m)$ is the set of methods referenced in the code of the method m (excluding recursive references).

That is, in order to apply the method m, an object of the class must have been created using an α-cut of the structure of the class with $\alpha \leq N_m$. Otherwise, the object would lack some necessary attributes to apply the given method.

- *In some particular situations the designer of the type can directly assign a degree to the method lower than the one deducted from the code (e.g., for a virtual or abstract method).*

3.1.3. Instantiation of Fuzzy Types

The change proposed in the concept of type involves modifications to the idea of instantiation. To create a new object of a given class, we must be able to choose the α-cut of properties of the type that will be used to represent it. To do that, the model has a generic method $new(\alpha)$ (with $\alpha \in (0,1]$), called *fuzzy constructor*. The receptor of this method can be any class C, whereas the parameter is the level α of the structure needed to represent the new object. The effect of sending the message $new(\alpha)$ to a class C with structural component S and behavior component B consists of creating an object incorporating the set S_α of attributes. The set B_α of methods defines the behavior of this object.

Let us use an example to clarify the previous definitions. Suppose that we want to define the concept of *Image*, and we use the following three levels of precision:

- *minimum features*: theme, file, format, version, age, quality
- *first level of precision*: horizontal resolution, vertical resolution, and palette
- *second level of precision*: histogram, frontiers, bands, and convolution.

The structure of the type Image can be expressed using the following fuzzy set: S = 1/theme + 1/file + 1/format + 1/version + 1/age + 1/quality + 0.9/hor_res + 0.9/ver_res + 0.9/palette + 0.8/bands + 0.8/histogram + 0.8/convolution + 0.8/frontiers.

This set has three relevant α-cuts:

- S_1 = {theme, file, format, version, age, quality}
- $S_{0.9} = S_1 \cup$ {hor_res, ver_res, palette}
- $S_{0.8} = S_{0.9} \cup$ {histogram, convolution, frontiers}.

In relation to the type Image, it is possible to find methods with a precision level 1 (as get_file(...), change_theme(...), etc.), with a precision level 0.9 (as change_palette(...), etc.) or even with a level 0.8, which generate image convolutions or histograms.

When the user of our type wants to create an instance of the class Image, he will be able to make it incorporate the attributes of either S_1, or $S_{0.9}$, or $S_{0.8}$, according to the precision degree required by this object to be represented.

3.1.4. Inheritance

The inheritance mechanism H must enable part of the class structure and behavior to be inherited by its subclasses. As we have done with the instantiation

mechanism, a threshold is added to indicate the properties we want to be inherited. Two different forms of inheritance can be considered:

- Incorporating inherited attributes and methods to the kernel set of the structural and behavioral components of the subclass, respectively. In this way, the fuzziness of the inherited properties is eliminated. This type of inheritance is called *inheritance without propagation* H_{crisp}.
- Keeping the fuzziness, by inheriting both properties and methods affected by the corresponding membership degree. This type of inheritance is called *inheritance with propagation* H_{fuzzy}.

Let us suppose that we are creating a new class C. We would like it to inherit from another class C' without propagation with a threshold α. S_{Sub} and B_{Sub} stand for the structural and behavioral components defined for the new class. S_{Sup} and B_{Sup} stand for the respective components of the superclass. The resulting type for the class C will have the following components:

$$S_C = S_{Sup_\alpha} \cup S_{Sub} \tag{5}$$

$$B_C = B_{Sup_\alpha} \cup B_{Sub} \tag{6}$$

However, in order to inherit with propagation, the new components would be:

$$S_C = S^*_{Sup_\alpha} \cup S_{Sub} \tag{7}$$

$$B_C = B^*_{Sup_\alpha} \cup B_{Sub} \tag{8}$$

In these last formulas, for any fuzzy set Y, Y^*_α stands for the fuzzy cut at level α of Y, defined by the following membership function:

$$\mu_{Y^*_\alpha}(m) = \begin{cases} 0 & \text{if } \mu_Y(x) < \alpha \\ \mu_Y(x) & \text{otherwise} \end{cases} \tag{9}$$

In both kinds of inheritance, B_{Sup_α}, $B^*_{Sup_\alpha}$, and B_{Sub} are calculated once the definitive structural component S_C is known.

Figure 3 depicts a hierarchy of classes related to the Image class (in the figure, $H_{c,\alpha}$ and $H_{f,\alpha}$ stand for H_{crisp} with threshold α and H_{fuzzy} with threshold α, respectively).

3.1.5. *Conventional Representation of Fuzzy Types*

This new way of considering the type definition can be easily modeled over a conventional object-oriented model, using the concept of 1-ramified hierarchy of classes.[16]

DEFINITION 4. *A 1-ramified hierarchy of classes is defined as a series of classes* $C_1, \ldots, C_{i-1}, C_i, C_{i+1}, \ldots, C_n$ *verifying the following properties:*

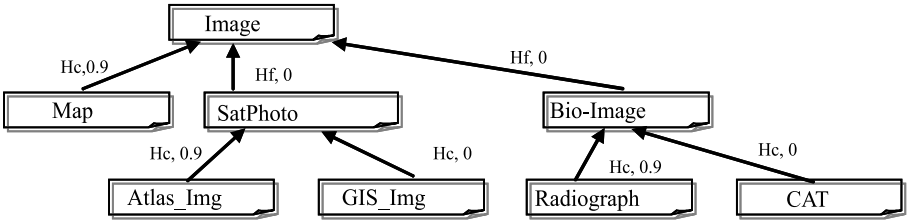

Figure 3. A hierarchy of fuzzy types.

- $Sub_{Ci} = \{C_{i+1}\}, i = 1 \ldots n-1$ (Sub_{Ci} *stands for the set of direct subclasses of* C_i*).*
- $Sup_{Ci} = \{C_{i-1}\}, i = 2 \ldots n$ (Sup_{Ci} *stands for the set of direct superclasses of* C_i*).*
- *A finite sequence of values* $\{\alpha_i\}$ *exists, associated to the hierarchy, such that* $\alpha_1 = 1$, $\alpha_n > 0$ *and* $\alpha_i > \alpha_{i+1}$ $(i = 1 \ldots n-1)$.

Figure 4 shows this idea for the class Image. The figure depicts the hierarchy of classes needed to support the type Image, where properties are ranked in three relevant α-cuts. Using this kind of hierarchy we can easily model the instantiation mechanism (we only have to chose the right class of the hierarchy and apply the conventional *new* method). The implementation of the two kinds of inheritance is a bit more complicated.[17]

3.1.6. The Meaning of α-Values

It may happen that the concrete value used for α might not be very important, if the only purpose is to organize a structure in a certain number of precision levels. In this situation, the important aim is the number of relevant α-cuts used. For example, a structure with three levels of precision can be obtained using both the set of values $\{1, 0.9, 0.8\}$ and $\{1, 0.99, 0.98\}$ (there are infinite possibilities). Nevertheless, in other situations, giving some semantics to these values may be necessary. For example, if the structure is being inferred from a set of instances, the values of α can be used to indicate not only a partition in precision levels of the type but also the relevance of each attribute within the type that is being defined. Not only will the number of precision levels be important, but the dispersion degree among these levels will be very interesting, too. In this kind of situation, the set of α values has an additional meaning, and the Fuzzy Set Theory is better exploited in the modeling of types.

3.2. Advantages of the Use of Fuzzy Types

Fuzzy types can be represented using conventional object-orientation, as we saw in the previous subsection. Using conventional inheritance we can solve the same problems as using fuzzy types. However, in designs like the one of Images that we described in the previous subsections, the use of fuzzy types reduces the complexity of the schema and eases the task of the designer. When using conventional object-orientation, the schema for the aforementioned example will look like the example in Figure 5.

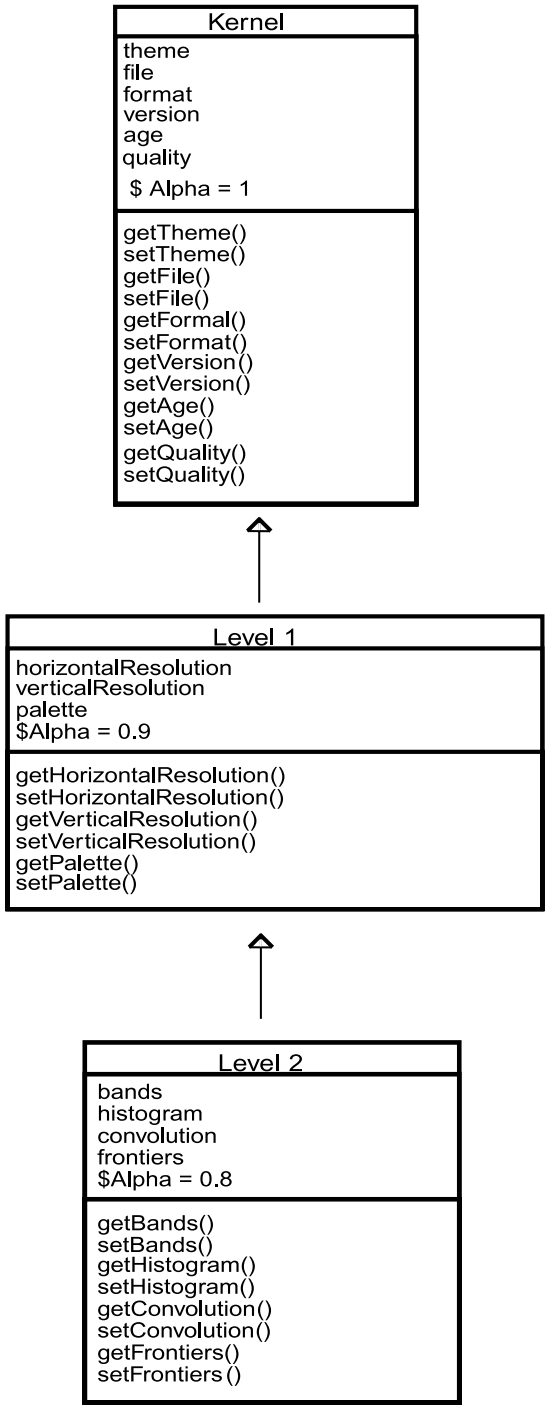

Figure 4. 1-ramified hierarchy for images.

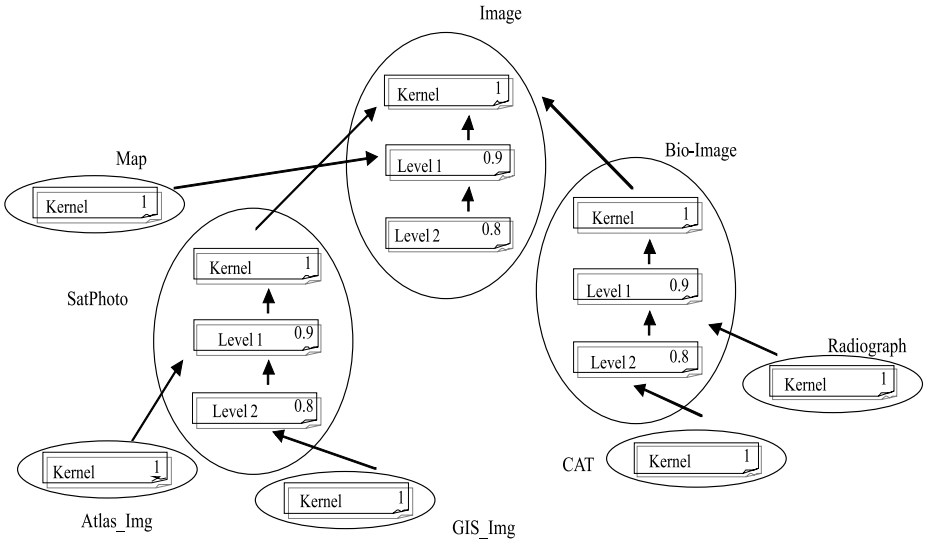

Figure 5. Conventional hierarchy for the images example.

The use of a fuzzy type to represent the structure of classes like in the example of Images permits us to keep unique types in the schema (which is better from the semantic point of view), avoiding null values. As we will see, Foodbi permits the designer to create a schema using fuzzy types that later are translated into the conventional inheritance hierarchy that supports them.

3.3. Attribute Level: Imprecise Domains and Fuzzy Collections

Let us now focus on the attribute values that constitute the object state. The capability of handling imperfect attribute values must be built according to the different semantic interpretation of the domains where these values are defined.

There exist models (i.e., Refs. 10 and 18–20) that make a clear distinction about conjunctive and disjunctive semantics of fuzzy attribute values. Labels whose semantics is a possibility distribution defined over a reference set of possible values have also been used by Bordogna et al.[12,21]

In our approach, we take as a basis these theoretical considerations and present a complete way to treat imperfect information at this level: we use the expressiveness of linguistic labels to set out imprecise values, but we consider the different semantics that these labels may have according to the features of the domain where they are defined.

3.3.1. Imprecise Attribute Domains

The reasons why an attribute value can be ill-defined may be assorted, from an actual ill-knowledge of the datum to an inherent imprecision affecting the domain of the attribute. Let us consider the following examples:

(a) The image is *old*.
(b) The quality of the image is *high*.

There exists an underlying domain below the label used in the first sentence (e.g., a range of numerical years). In contrast, it is not easy to find such underlying domain for the label *high* of the second sentence. Therefore, when defining domains constituted by labels, we have to distinguish between those cases with and without semantic representation on an underlying domain.

- *Imprecise values without semantic representation*. This case brings together all the examples similar to the sentence (b): *The quality of the image is high*. In those situations, the domain of the attribute is a set of linguistic labels (e.g., very low, low, medium, high, very high). We cannot define the semantics of the labels by means of fuzzy sets built over an underlying domain. Thus, we cannot use the classical equality in order to compare two labels, and we have to use a different comparison schema. Our proposal has its basis[13] on the use of a resemblance relationship in order to adequately handle the inherent imprecision of the labels set. This relationship allows us to compare the labels, generalizing the basic equality criterion.

The domain for handling labeled image qualities could be as shown in Table I.

- *Imprecise values with semantics representation*. In those cases (sentence (a): The image is *old*), the attribute has a well-defined basic domain (usually a bounded subset of the real interval) and the labels that stand for ill-defined values can be easily described by means of fuzzy subsets defined over the aforesaid basic domain.

However, we have to be careful with the semantics assigned to these fuzzy sets, because we can have two possible situations:

- The actual value is one of the values of the support set. In this case the fuzzy set is interpreted in a disjunctive sense (as a possibility distribution of values).
- The actual value is a combination of the values of the support set. In this case the fuzzy set is interpreted in a conjunctive sense (as a set where membership is gradual).

Let us first focus on the first case. From a structural point of view, we have an attribute with an atomic domain constituted by labels. However, in contrast to sentence (b), the labels have a semantics and the resemblance relationship must be built as an extension of the classical equality that holds in the underlying domain.

Table I. Quality labels and their similarity.

	Very low	Low	Medium	High	Very high
Very low	1.0	0.7	0.5	0.2	0.0
Low		1.0	0.7	0.5	0.2
Medium			1.0	0.7	0.5
High				1.0	0.7
Very high					1.0

Let a be an attribute with a basic domain B that takes values either in B or in a set of labels L whose semantics is expressed by means of fuzzy subsets of B ($\forall l \in L, \exists \mu_l : B \rightarrow [0,1]$). We have to follow the next steps in order to manage the final attribute domain D:

- To build $D = B \cup L$.
- To define a resemblance relation S in D so that $\forall x, y \in D$:

$$\mu_S(x,y) = \begin{cases} 1 & (x = y) \wedge (x, y \in B) \\ 0 & (x \neq y) \wedge (x, y \in B) \\ \mu_l(z) & ((x = l \in L) \wedge (y = z \in B)) \\ & \vee ((y = l) \wedge (x = z \in B)) \\ \max\limits_{z \in B} (\mu_x(z) \otimes \mu_y(z)) & \text{otherwise} \end{cases} \tag{10}$$

Although Equation (10) is an accepted resemblance approach for these situations, there are several alternative proposals in the literature that describe a calculus for similarity and resemblance measures between fuzzy sets. A complete study of them can be found in Ref. 22.

3.3.2. Conjunctive Meaning: Fuzzy Collections

The previous lines have been devoted to dealing with atomic domains, domains constituted by atomic labeled values. But we have not studied the case where the actual value represented by the label is a combination of the values of the support set of the semantics of the label. In this case the fuzzy set is interpreted in a conjunctive sense (as a set where membership is gradual). That is, we have not considered the case where the value is not atomic but a fuzzy collection of values.

In general, it is not necessary to use a label to represent this kind of conjunctive value. In any case, if we want to represent these values in our system, we need suitable operators to compare them, taking into account that, now, the semantics of the fuzzy set is conjunctive. In Ref. 23, we present a set of operators to compare fuzzy collections of objects. Let us briefly describe them.

Conjunctive fuzzy set comparison is often done by means of the concept of inclusion:

$$A = B \text{ if, and only if, } (A \subseteq B) \wedge (B \subseteq A) \tag{11}$$

The inclusion degree of a fuzzy set A in a fuzzy set B can be computed as follows[9]:

$$N(B|A) = \min_{u \in U} \{I(\mu_A(u), \mu_B(u))\} \tag{12}$$

where I is a fuzzy implication operator and U is the reference set where A and B are defined.

Taking into account that we are in an object-oriented context, where objects are usually complex, the elements of U may be fuzzily described objects, in turn. In this case, 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.

To perform comparison among this kind of fuzzy collections, we propose the following set of operators:

- An inclusion operator (Θ), which takes into account resemblance between the elements that are being compared ($\otimes$ stands for a t-norm):

$$\Theta_S(B|A) = \min_{x \in \mathcal{U}} \max_{y \in \mathcal{U}} \theta_{A,B,S}(x, y) \tag{13}$$

  where

$$\theta_{A,B,S}(x, y) = \otimes\,(I(\mu_A(x), \mu_B(y)), \mu_S(x, y)) \tag{14}$$

- A generalized resemblance operator ($\sqsupset$), which considers both inclusion directions and which can be weighted multiplying by a cardinality ratio (Φ) ($\otimes$ stands for a t-norm):

$$\sqsupset_{S,\otimes}(A, B) = \otimes\,(\Theta_S(B|A), \Theta_S(A|B)) \tag{15}$$

$$\Phi(A, B) = \begin{cases} 1, & \text{if } A = \emptyset \wedge B = \emptyset \\ \dfrac{\min(|A|, |B|)}{\max(|A|, |B|)}, & \text{otherwise} \end{cases} \tag{16}$$

3.3.3. *Conventional Representation: A Supporting Framework*

The implementation of the model includes some conventional general classes (see Figure 6) to represent these common kinds of domains for imprecision, such as linguistic labels without underlying representation (*DomainWithoutRepresentation*), domains where labels are possibility distributions over an underlying basic domain (*DisjunctiveDomain* and its subclasses to represent labels with finite support set, basic domain values, and functional representations of labels with infinite support set, like trapezoidal ones), and, finally, fuzzy collections of fuzzy objects (*ConjunctiveDomain*).

For example, to create the Age domain for images, Foodbi uses the classes depicted in Figure 6 under the DisjunctiveDomain class. Taking this into account, the translation performed by Foodbi is reduced to the generation of the following Java code:

```
public class Age extends AbstractDisjunctiveDomain {

 // Redefined constructor

 public Age (int basicValue) {
  super( new Integer(basicValue) );
 }
```

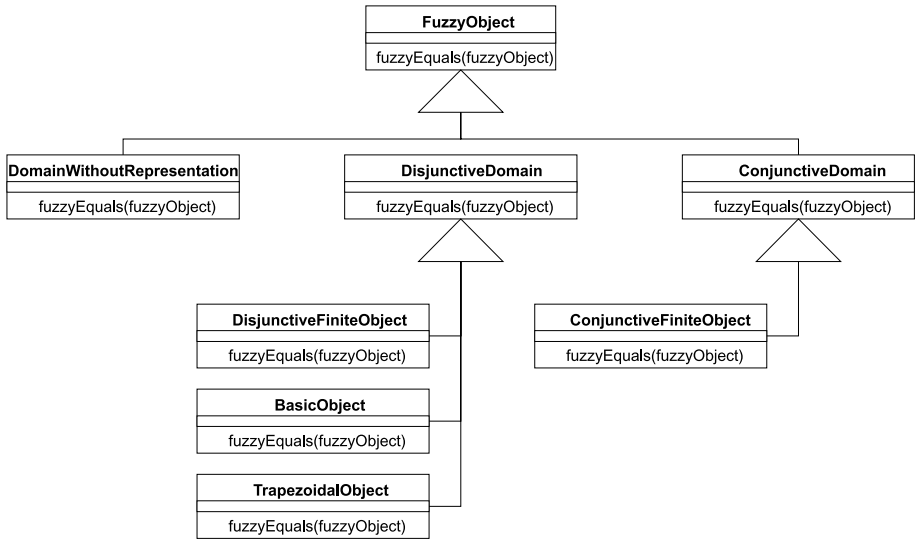

Figure 6. The underlying Java framework.

```
 public Age (Label label, Vector semantic) {
  super (label, semantic, Integer.class);
 }
 public Age (Label label, float a, float b, float c,
  float d) {
  super (label, a, b, c, d);
 }
}
```

In this way, not only is the translation performed by Foodbi not complex, but also, if the user does not want to use Foodbi as an interface, he can easily develop the code mentioned above.

3.3.4. FuzzyEquals: The Basis for Queries

Actually, the framework that supports imprecision management in Foodbi also includes comparison capabilities that serve as the basis for queries.[23,24] The framework includes a FuzzyEquals method that is overridden all over the hierarchy and is able to compare every couple of objects of a given class. This method is implemented using reflection, and users do not have to implement it in their classes:

- Reflection is a feature of many of the modern programming languages (e.g., Java—java.sun.com—and C#—www.microsoft.com). This feature allows an executing program to examine or "introspect" itself and manipulate internal properties of the program.

For example, it is possible for a class to obtain the names of all its members and display them.

- Because our final aim is to allow the programmer to perform fuzzy comparisons without having to write specific code for each class she writes, we can define a generic *FuzzyObject* class that will serve as a basis class for any classes whose objects need fuzzy comparison capabilities. We can avoid duplicating code in different classes if we write a generic *fuzzyEquals* method in the *FuzzyObject* class. Taking into account that the *fuzzyEquals* method requires access to the particular object fields, the only way we can implement such a general version of this operator is through reflection.

Just by extending this general *FuzzyObject*, the programmer (or Foodbi) can define her own classes to represent fuzzy objects.

FuzzyEquals has different implementations in the framework:

- In classes that represent the different kinds of imprecise domains, FuzzyEquals is implemented using the operators that we have described previously.
- In FuzzyObject class, FuzzyEquals computes a deep comparison of objects. First, resemblance information is collected by studying every couple of attribute values, and then, this information is aggregated and a general resemblance opinion for the whole object is computed. This process is expensive in terms of computing because in complex objects it implies recursive computation and, frequently, the same comparison is computed more than once. In the framework, this cost is attenuated by keeping a pool of recent comparisons and applying some techniques of approximation when needed. A deep study of the comparison technique can be found in Ref. 24.

3.4. Additional Structural Enhancements

Foodbi also permits us to use some additional structural enhancements, as well as fuzzy types, imprecise domains, and fuzzy collections. Among them are:

- **Explicit uncertainty in attribute values.** In the previous subsections, we have studied the representation of imprecise attribute values of different kinds. Nevertheless, as we say in the introduction of this article, there exists a close relationship between imprecision and uncertainty.

  There may be situations where, as well as an implicit uncertainty, we have to deal with an explicit uncertainty. For example, consider the following sentences:

  (a) It is *sure* that the image is old.
  (b) It is *very possible* that the image name is biotech.

  Different measure scales for the uncertainty of the attribute value may be considered: we can use probability (possibility) measures defined within the [0,1] interval, linguistic labels of probability (possibility) whose semantic representation is a disjunctive fuzzy set, certainty measures, evidences, and so forth.

  According to the above paragraphs, to consider that some kind of uncertainty is associated to some attribute implies assuming that the attribute domain is an aggregation of the actual attribute domain and the scale where this lack of certainty is measured.
- **Semantically enhanced relationships among objects.** In some schemata, it may be interesting to weight the connection among objects (expressed by an attribute of a given class) by means of a degree of strength. In these situations, as suggested by Bordogna et al.,[12] we can use numerical or linguistic values to express this strength. For example, we can consider the set {"high," "medium," "low"} of labels, with each label represented as a disjunctive fuzzy set in [0,1].

Notice that the strength has a semantic interpretation different from the one given to uncertainty. In this case, we know that the objects are related, but we consider different strengths in their connection.

- **Fuzzy class extents.** When designing the schema of a given application, many reasons may demand the use of fuzzy extents in the classes, not from a structural point of view (fuzzy types and fuzzy instantiation) but from a semantic point of view. In this case, our application semantics may require the gradual expression of the object membership to the class it belongs to. In many cases, these needs appear connected to the presence of imperfection in the attribute values that characterize the objects.

  The membership degree is normally valued within the [0,1] interval, changing the set of objects that constitute the class (i.e., the class extent) into a fuzzy set of objects.

  There are many proposals in the literature that suggest different ways in which the membership degree of a given object to a certain class can be computed, in case this membership degree must be inferred from its attribute values in relation to the archetypical or expected ones for the class. Most of these approaches are founded on concepts such as *inclusion* or *typicality*.[9,12,18]

  The representation of this feature in a classical object-oriented model is simple: we only have to add an extra attribute to the class we want to extend in a fuzzy way. The domain of this attribute could be
  - the interval [0,1]
  - a set of linguistic labels that expresses membership, defined over the aforementioned interval.

The designer of the type is responsible for implementing the way the membership degree is computed (usually, by means of a method). In case the degree depends on the values of some attributes, the user of the type is responsible for invoking the method that gets the value of the degree when the object changes.

4. CREATING THE *IMAGE* CLASS WITH FOODBI

As we have previously mentioned, Foodbi is a graphical user interface that allows the management of fuzzy object-oriented databases. Using Foodbi, the user can build a hierarchy of classes with fuzzy types, where attributes can be valued over suitable domains for expressing fuzziness.

The current implementation of Foodbi uses the following technology:

- the Java object-oriented programming platform[14]
- the Oracle object-relational DBMS to provide persistency for the object-oriented schemata developed in Java.

The core part of Foodbi is devoted to facilitating the creation of classes with extended characteristics. When defining a new class, the user is asked for the following information:

- general metadata that describe the class: identifier, kind of extent (crisp or fuzzy), description, and so on
- set of attributes that characterize its structural component (which can also be a fuzzy set)
- set of methods that constitute its behavioral component (which can also be a fuzzy set)
- a model of inheritance, using the fuzzy inheritance extensions we discussed above.

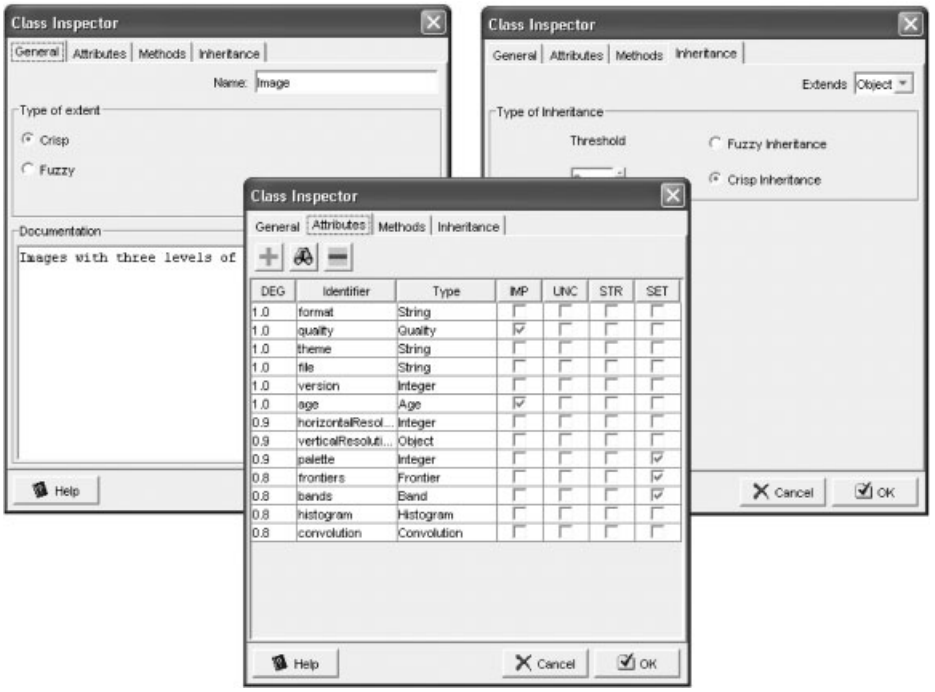

Figure 7. The *Image* class in Foodbi.

Figure 7 illustrates the Foodbi class inspector when we are defining the *Image* class.

The information provided by the user when defining the attributes for the *Image* class determines the way in which fuzziness will be handled:

- Foodbi collects general metadata about the attribute: identifier, membership degree, visibility, and so on.
- In the case of attributes with imprecise values, the user can build labeled domains by choosing among different semantics: with or without underlying basic domain, disjunctive, conjunctive, and so on.
- In case the attribute value can be affected by explicit uncertainty, the user can attach a set of linguistic labels to the attribute domain (or the [0,1] interval).
- The user can even graduate the relationship expressed by the attribute, combining the attribute domain with a suitable linguistic domain for expressing strength values.

For example, Figure 8 illustrates the creation of the *Quality* attribute quality for the *Image* class, whereas Figure 9 shows the creation of the *Age* attribute.

Foodbi includes a set of wizards for assisting the user in the creation of new exemplars of the different kinds of domains we described in the previous section.

Method definition is performed as in any other object-oriented platform: the user specifies the method interface and then implements its code. Foodbi can

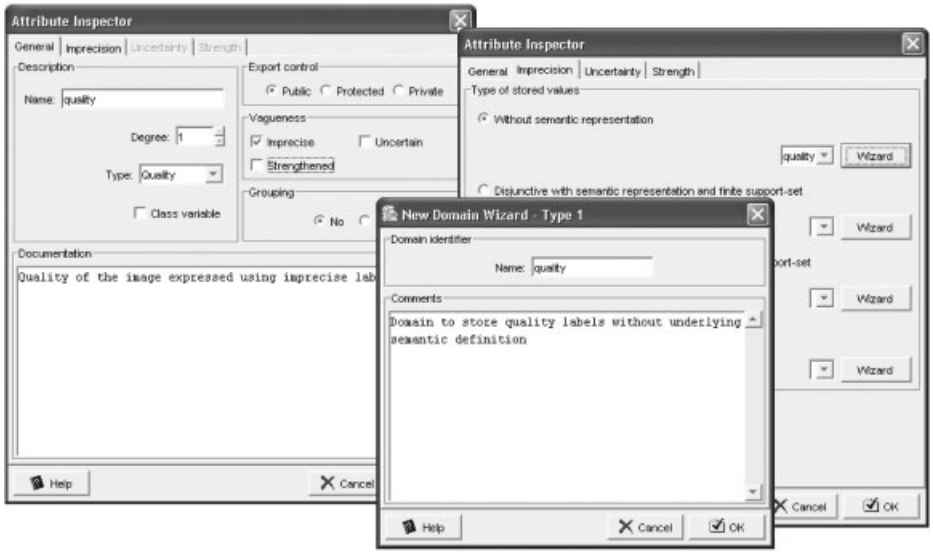

Figure 8. The *Quality* attribute in Foodbi.

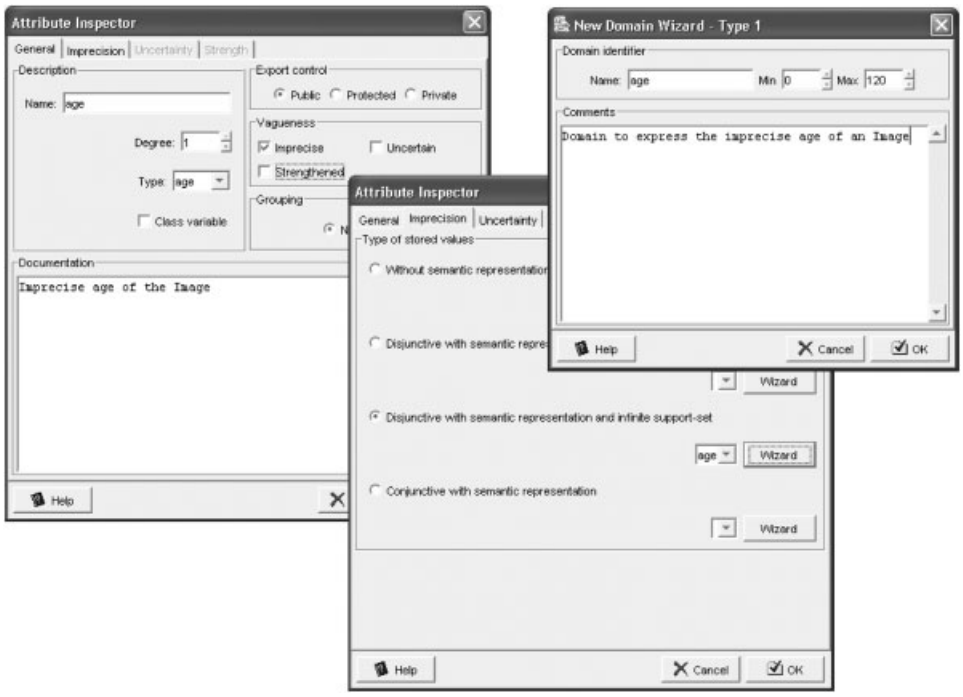

Figure 9. The *Age* attribute in Foodbi.

Table II. Files generated by Foodbi for the *Image* class.

File	Description
Quality.java	File with definition of the *Quality* imprecise domain
Age.java	File with definition of the *Age* imprecise domain
Image.java	File with the kernel of the *Image* class
Image_90.java	File with the first precision level for the *Image* class
Image_80.java	File with the second precision level for the *Image* class

automatically generate accessor and modifier methods for the set of attributes that constitute the type being defined.

Once the class description is completed, Foodbi translates it into a set of standard Java classes that implement it, just by following the guidelines explained in the previous section.

Once the *Image* class is created and its attributes are defined, Foodbi automatically generates the set of Java files listed in Table II.

Following the same process, we can create the whole hierarchy shown in Figure 3 and Foodbi will generate the set of files depicted in Figure 10.

5. CONCLUSIONS AND FURTHER WORK

In this article, we have presented an architecture for the development of a fuzzy object-oriented database management system. We have studied several

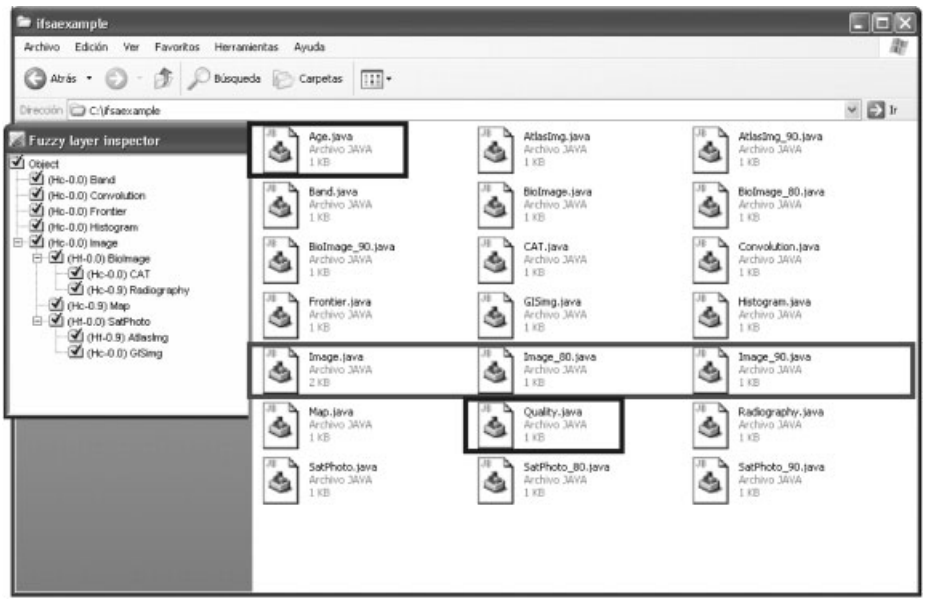

Figure 10. Automatically generated code for the class hierarchy shown in Figure 3.

suitable strategies to address the representation of different kinds of imperfections that may arise when a database is designed following the object-oriented paradigm. Because the new structures needed to support data imperfection are implemented using standard object-oriented techniques, we can use existing conventional database systems as the underlying infrastructure for fuzzy object-oriented database systems.

One of the topics that remains unaddressed is the modeling of the object behavior when vagueness affects the values of its properties. Though some approaches have been proposed in the literature,[5,25] we think that this topic still deserves further study, maybe in the context of hypothetical reasoning.

The Foodbi prototype is currently being used as the basis for two main related efforts:

- the completion of a complete fuzzy object-oriented data management system
- the development of a general object-oriented class library, which can be used to manage fuzzy information without the need of any additional resources.

Acknowledgment

This article has been supported in part by the Spanish "Comisión Interministerial de Ciencia y Tecnología" under grants TIC2003-08687-C02-02, TIC2002-00480, and TIC2002-04021-C02-02.

References

1. Zadeh LA. Fuzzy sets. Inform Contr 1965;8:338–353.
2. De Caluwe R. Fuzzy and uncertain object-oriented databases: Concepts and models. Singapore: World Scientific; 1997.
3. Lee J, Kuo JY, Xue NL. A note on current approaches to extend fuzzy logic to object oriented modeling. Int J Intell Syst 2001;16:807–820.
4. Gonzalez A, Pons O, Vila MA. Dealing with uncertainty and imprecision by means of fuzzy numbers. Int J Approx Reason 1999;21:233–256.
5. Van Gyseghem N, De Caluwe R. Imprecision and uncertainty in the UFO database model. J Am Soc Inform Sci 1998;49:236–252.
6. Ruspini EH. Imprecision and uncertainty in the entity-relationship model. In: Prade H, Negoita CV, editors. Fuzzy logic and knowledge engineering. Cologne, Germany: Verlag TUV Rheinland; 1986. pp 18–28.
7. Zivieli A, Chen PP. Entity-relationship modeling and fuzzy databases. In: Proc Second Int Conf on Data Engineering—IEEE; 1986. pp 18–28.
8. Vanderberghe RM, De Caluwe R. An entity-relationship approach to the modeling of vagueness in databases. In: Proc ECSQAU—Symbolic and Quantitative Approaches to Uncertainty; 1991. pp 338–343.
9. Rossazza JP, Dubois D, Prade H. A hierarchical model of fuzzy classes. In: Fuzzy and uncertain object-oriented databases. Concepts and models. Singapore: World Scientific: 1998. pp 21–61.
10. George R, Buckles BP, Petry FE. Modeling class hierarchies in the fuzzy object-oriented data model. Fuzzy Set Syst 1993;60:259–272.
11. Koyuncu M, Yazici A. Ifood: An intelligent fuzzy object-oriented database architecture. IEEE Trans Knowl Data Eng 2003;15:1137–1154.

12. Bordogna G, Pasi G, Lucarella D. A fuzzy object oriented data model for managing vague and uncertain information. Int J Intell Syst 1999;14:623–651.
13. Marín N, Blanco IJ, Pons O, Vila MA. Softening the object-oriented database-model: Imprecision, uncertainty, and fuzzy types. In: Proc IFSA/NAFIPS World Congress; 2001. pp 2323–2328.
14. Spell B. Professional Java programming. Chichester, UK: Wrox Press, 2000.
15. Marín N, Vila MA, Pons O. Fuzzy types: A new concept of type for managing vague structures. Int J Intell Syst 2000;15:1061–1085.
16. Marín N, Vila MA, Pons O. A strategy for adding fuzzy types to an object-oriented database system. Int J Intell Syst 2001;16:863–880.
17. Marín N, Vila MA, Blanco IJ, Pons O. Fuzzy types as a new layer on an object oriented database system. In: Proc IPMU, Madrid, Spain; 2000. pp 1099–1106.
18. Yazici A, Aksoy D, George R. The similarity-based fuzzy object-oriented data model. In: Proc IPMU, vol 3; 1996. pp 1177–1182.
19. Yazici A, George R, Aksoy D. Design and implementation issues in the fuzzy object-oriented data model. J Inform Sci 1998;108:241–260.
20. Yazici A, Koyuncu M. Fuzzy object-oriented database modeling coupled with fuzzy logic. Fuzzy Set Syst 1997;89:1–26.
21. Bordogna G, Lucarella D, Pasi G. A fuzzy object oriented data model. In: Proc FUZZ-IEEE; 1994. pp 313–317.
22. Zwick R, Carlstein E, Budescu DV. Measures of similarity among fuzzy concepts: A comparative analysis. Int J Approx Reason 1987;1:221–242.
23. Marín N, Medina JM, Pons O, Sánchez D, Vila MA. Complex object comparison in a fuzzy context. Inform Softw Technol 2003;45:431–444.
24. Berzal F, Cubero JC, Marín N, Pons O. Enabling fuzzy object comparison in modern programming platforms through reflection. In: Bilgiç T, de Baets B, Kaynak O, editors. Fuzzy sets and systems. Berlin: Springer-Verlag; 2003. pp 660–667.
25. Van Gyseghem N, De Caluwe R. Fuzzy object-oriented databases: Some behavioral issues. In: Proc EUFIT; 1994. pp 361–364.