Fuzzy object-oriented database modeling coupled with fuzzy logic

Adnan Yazici*, Murat Koyuncu

Department of Computer Engineering, Middle East Technical University, Ankara 06531, Turkey

Received September 1995; revised February 1996

Abstract

This paper presents a modeling approach which couples fuzzy object-oriented database modeling with fuzzy logic. The modeling approach introduced here handles fuzziness at attribute, object/class and class/superclass levels in addition to fuzziness in class/class relationships and various associations among classes. We utilize logical rules to define some of the crisp/fuzzy relationships and associations which cannot be presented easily with object-oriented modeling features alone in the class hierarchies. We think that incorporation of object-oriented database modeling with logic along with usage of fuzzy set theory simplifies the design of complex and knowledge-intensive applications and handles uncertainty effectively, therefore resulting in a powerful modeling framework. © 1997 Elsevier Science B.V.

Keywords: Object-oriented database modeling; Uncertainty; Fuzzy set theory; Logic programming

1. Introduction

In recent years, the object-oriented database model has gained a popularity, which provides powerful data modeling features. This modeling approach [4, 12, 37] is very adequate in developing database management systems for new and emerging applications including CAD/CAM, Office Automation Systems, Engineering Designs, etc. On the other hand, the deductive database model [18, 28] is another modeling approach towards the development of new generation database systems.

While these two models are being developed separately, there have been also emerging researches for developing deductive object-oriented database models [1, 5, 8, 14, 17, 22, 25].

Most of the existing database models including the deductive object-oriented database models are designed under assumptions of precision of both the data stored in the databases and the requests to retrieve data. That is, anything uncertain must be precisely defined; therefore, anything uncertain must be either precluded or made artificially precise. In real life, this assumption of precise world is often not appropriate for many complex applications (i.e. Office Automation Systems, Decision Support Systems, Geographic and Environmental Information Systems, Expert Database Systems), since they involve in not only complex data/knowledge but also some kinds of uncertainty. When precise information is unavailable about the...
miniworld, it is often the case that some relevant information is available. In these cases, it may be advantageous to design methods for impreciseness by which this information can be stored, manipulated and retrieved. Uncertainty may also be present in requests to retrieve data, when users formulate their queries with imprecise terms.

Uncertainties are usually in the form of null, range, and descriptive form which is also referred to as fuzzy [24, 26, 36]. Since the complex applications mostly involve in qualitative information of the experts of the subject, we will be primarily focusing on handling fuzziness in this study. Some attempts have already been made to represent and manipulate fuzzy data in various database models such as the relational database model [7, 26, 30], the NF² database model [30, 31], the object-oriented database model [9, 11, 12, 16, 29, 30], and in some knowledge-based systems [27, 31]. In general, existing data models are extended either utilizing similarity [7, 11, 30] or possibility [9, 26] theories. More specifically, Dubois et al. [9] used possibility theory for representation of vagueness and uncertainty in class hierarchies. They define the inclusion of classes to be the inclusion between their fuzzy ranges in possibility distribution form for single-valued attributes. In this way they formulate the certainty of membership of an object in a class. In later studies, George et al. [11] and Aksoy and Yazici [2, 3] have studied similarity-based fuzzy object-oriented database models in which they consider both fuzziness in attribute values and in classification. An object-oriented data model that represents uncertainty in addition to fuzziness has been proposed by Gyseghem et al. [29], in which uncertainty is handled by generalized fuzzy sets and fuzzy information by conjunctive fuzzy sets. Inoue et al. [16] studied “Fuzzy Set Object” as a first-class object in the programming language with the aim of developing both hardware and software for a fuzzy computer system to process vague information. Fuzzy Objects have also been studied by Graham [12], in which objects are extended for handling fuzziness in two ways: First, objects are allowed to contain fuzzy sets as their attribute values which can be inherited. Second, inheritance is allowed to be partial; that is, fuzzy objects may inherit both crisp and fuzzy attribute values partially, by using a defined certainty value.

In this article we introduce a modeling approach of which the fuzzy object-oriented database model is tightly coupled with fuzzy logic [35]. The fuzzy object-oriented database model (FOOD) utilized here can be considered as an extension of the model (GBP) introduced in [11]. One of the important differences between FOOD used in this study and the GBP model is that the FOOD model uses fuzzy values only when precise information is unavailable; that is, we handle fuzzy and crisp values together for the same attribute of an object by using manipulation of membership values. The other important difference is that the FOOD model handles fuzziness at attribute level, object/class level and class/superclass level with an enhanced description; thus, leading to a truer representation of fuzzy hierarchies. The description along with the detailed examples of the FOOD model will be given later in the next section.

The main task of this paper is not to describe the FOOD model, but mainly to introduce how to define uncertain relationships and associated constraints among classes in a class hierarchy by tightly coupling fuzzy logic with object-oriented database modeling for which the FOOD model is utilized. We argue that our approach of coupling fuzzy logic with the FOOD model results in powerful modeling framework for handling complex and uncertain applications mentioned above. In order to show the modeling power of our approach, we apply it to an application, namely Environmental Information System, by implementing it on POPLOG environment. The implementation includes a class hierarchy along with a set of rules and facts to define the complex and uncertain relationships and associated constraints among classes. The fuzzy rules and facts are stored in a knowledge-base and the fuzzy objects are stored in a database and some utilities are developed for tightly coupling the database and the knowledge-base. Thus, many possible uncertain queries could be formulated and the answers to these queries could be retrieved from this system effectively. A user interface is also developed by utilizing X Windows with Open Look widget set.
In this paper, we first introduce the fuzzy object-oriented database (FOOD) model for incorporating fuzziness at attribute, object/class and class/superclass levels after describing our application example namely Environmental Information System (EIS). Then we state our reasons for coupling fuzzy logic with fuzzy object-oriented database modeling. The important feature of our model involving in the usage of fuzzy logic coupled with the object-oriented database modeling for handling complex knowledge, crisp/fuzzy relationships and associated constraints among classes is included in Section 5. In Section 6, the implementation of our modeling approach applied to EIS application is briefly described before we state the conclusions.

2. An application: environmental information system

In recent years, environmental pollution has become an important problem in the world. An environmental information system usually holds and evaluates complex data about the environment. Since much of the data also involves in uncertainty, existing data models have difficulties in providing meaningful judgments and therefore solutions to the problem. Some properties of the environment system, for instance, the location, the volume, and the effects of pollutants to various sites, are very often not precisely known.

A possible representation for an environmental information system is given in Fig. 1. In the figure, OBJECT is the common superclass as defined in all object-oriented database systems. In general and for simplicity, we deal with two main class definitions, the pollutants and the sites, with corresponding attributes specified later. Each of these two classes has its own subclasses. What we want to find out is the effects of pollutants, if it exists, to the various important sites. The following queries are such examples which are relevant for such a system:

“Find the pollution which affects the forest F1”
or
“Find the water-supplies affected by the pollutant P1”

where F1 is an object of the class forest and P1 is an object of the class pollutants. To answer these types of queries, we must define a relationship among the classes pollutants and sites.

Mostly information obtained for some pollution may be fuzzy and we still need to decide whether there is any pollution or not based on imprecise information. If pollution exists, we need to find out its effects. Different sites may be affected with different degree depending on the pollutants. For example, the effect of a pollutant, i.e. inorganic chemicals, to a forest and a water supply may be different. The pollution of some sites may be more important than the others, i.e. arsenic-polluted water supply is much more urgent than polluted forest. Maybe the pollution caused by each
contaminant is not so important, but two or more chemical contaminants found in the same site together cause some more serious pollution. We could increase such examples of relationships and associated constraints among the classes for this application. From all these, one may want to determine which location of a site needs to be more closely examined further to find out accurate effects of the contaminant(s) to the site. We could increase the number of possible important requests from such a system.

3. The fuzzy object-oriented database model

Before we proceed further we should give some background information related to the fuzziness and the fuzzy object-oriented database model at this point. Fuzzy logic [33] which is closely connected to fuzzy sets (proposed by Prof. L.A. Zadeh in 1965 [31]) is a generalization of the classical two-valued logic. The outstanding feature of fuzzy sets is the ability to express the amount of ambiguity in human thinking and subjectivity in a comparatively undistorted manner. Fuzzy set theory has proved its power in handling uncertainty in many commercial expert systems, control systems and many other products [13,34]. A fuzzy subset $F$ consists of objects $u \in U$ (where $U$ is the universe of discourse) is characterized by a membership function $\mu_F$, where $\mu_F: U \rightarrow [0,1]$. $\mu_F(u)$ expresses the degree of membership of elements $u \in U$ in the fuzzy subset $F$. Example membership functions of fuzzy sets for “tall”, “average” and “short” are given in Fig. 2. Detailed description of fuzzy set theory and related concepts can be found in [30–36].

The basis of the fuzzy object-oriented database model is the replacement of equality with a similarity relation. For each fuzzy attribute, a fuzzy domain and a similarity relation [34] are defined. A similarity relation is a fuzzy relation which indicates the strength of the relation between every pair of the elements in a domain $D_i$ and satisfies the following conditions:

$$\begin{align*}
\text{if } x, y, z \in D_i \text{ then } & \mu_s(x,x) = 1 \quad (\text{reflexivity}), \\
\mu_s(x, y) &= \mu_s(y, x) \quad (\text{symmetry}), \\
\mu_s(x, z) &\geq Max\{Min[\mu_s(x, y), \mu_s(y, z)]\} \quad (\text{transitivity}).
\end{align*}$$

The concept of similarity relation is a generalization of the concept of equivalence relation. That is, the identity relation is replaced by the explicitly declared similarity relation of which the identity relation is a special case. A similarity relation $\mu_s(x, y)$, for a given domain, $D_i$, is a mapping of every pair of elements in the domain onto interval $[0,1]$. The domain and the similarity relation of the fuzzy attribute temperature is given in Table 1, as an example.

![Fig. 2. Membership graph of height.](image)
The fuzziness may occur at three different levels in our fuzzy object-oriented database model. These are the attribute level, the object/class level and the class/superclass level. A detailed description along with the examples of how fuzziness is handled in each level is given in the subsections below.

Similarity matrices are used to represent the relation within the fuzzy attributes. The domain, \( dom \), is the set of values the attribute may take, irrespective of the class it falls into. The range of an attribute, \( rng \), is the set of allowed values that a member of a class, i.e. an object, may take for an attribute. In general \( rng \subseteq dom \). For instance, assume that \( height \) is an attribute and the domain of height is between 0 and 230 cm. If there exists a class \( Student \), the range of height for the class may be 80 to 230 cm. A range for each attribute of the class is defined as a subset of a fuzzy domain. The range definition for attribute \( a_i \) of class \( C \) is represented by the notation, \( rng_c(a_i) \), where \( a_i \in Attr(C) \).

Similar objects are grouped together to form a class and fuzziness at object/class and class/superclass relations are examined. The membership of an object \( o_j \) in class \( C \) with attributes \( Attr(C) \) can be formulated. The idea of fuzziness is extended in the relation of an object with the class of which is created as an instance. An object belongs to a class with a degree of membership. Based on the considerations of relevance and range of attribute values, the membership of object \( o_j \) in \( C \) can be defined as

\[
\mu_C(o_j) = g[f(RLV(a_i, C), INC(rng_c(a_i)/o_j(a_i)))],
\]

where \( RLV(a_i, C) \) indicates the relevance of the attribute \( a_i \) to the class \( C \), and \( INC(rng_c(a_i)/o_j(a_i)) \) denotes the degree of inclusion of the attribute values of \( o_j \) in the formal range of \( a_i \) in the class \( C \). The degree of inclusion, determines the extent of similarity between a value (or a set of values) in the denominator with the value (or a set of values) in the numerator. The function \( f \) represents the aggregation over the \( n \) attributes in the class and \( g \) reflects the type of link existing between an instance (object) and a class/superclass (\( f \) and \( g \) are functions that may be inherited from the superclass or may be defined within the local class). The value of \( RLV(a_i, C) \) may be supplied by the user or computed in a manner similar to that of [9]. Several cases are possible for the evaluation of \( INC(rng(a_i)/o_j(a_i)) \). A description along with the examples of how fuzziness is handled in each level is given in the subsections below. Interested readers can refer to \([3,11]\) where these cases are described in greater detail.

### 3.1. Attribute level

At the attribute level, there are different types of uncertainty of the attribute values. The uncertainty type primarily considered in this study is that information is available, but in descriptive term, in the absence of precise data. Such uncertain data is referred to as fuzzy. For example, a value of the \( height \) attribute which is \( tall \) is a fuzzy value. Fuzzy attributes may take a set of fuzzy values having one of the \( AND, OR \) and \( XOR \) semantics. The notation and the interpretation of these semantics are as follows:

**AND semantics:**

Notation: \( pollution.contaminant = \langle \text{SO}_2, \text{H}_2\text{S} \rangle \)

Interpretation: “Pollution includes both the contaminant \text{SO}_2 and \text{H}_2\text{S}”.

**OR semantics:**

Notation: \( ankara.temperature = \{ \text{normal, mild} \} \)

Interpretation: “Temperature of Ankara is normal or mild or both. Some people say that it is normal, some say that it is mild and some say it is normal and mild at the same time”.

**XOR semantics:**

Notation: \( deniz.sex = [\text{male, female}] \)

Interpretation: “Sex of Deniz is either male or female, but we do not know actually which one”.

The appropriate connection operator (\( AND, OR, XOR \)) must be used according to the meaning given to the attribute values. Otherwise, the interpretation of attribute values may be meaningless. It is possible to say that \( OR \) operator is the most fitting operator to the nature of the fuzziness. It holds more fuzziness in its semantics. The formal representation and description of all these operators can be found in \([3,21]\).

If we know the precise value of a fuzzy attribute, we use that value. Therefore, in this model we
handle both crisp and fuzzy values for attributes. Because of that, some objects may have fuzzy values, some may have crisp values for the same attribute. It is also possible to query the database both with fuzzy conditions and with crisp conditions. In order to handle both crisp and fuzzy values uniformly, the membership functions are used to calculate the membership degree of crisp values to determine the corresponding fuzzy set which the crisp values belong to. For example, the following function is used for the fuzzy attribute temperature:

\[
\mu_F(x) = \exp\left(-\frac{1}{2}\left(\frac{x - m}{\sigma}\right)^2\right),
\]

where \(x\) is the crisp value, \(m\) is the central value, \(\sigma\) is the spread of the fuzzy term and \(\mu\) is the membership value of \(x\) to the fuzzy set \(F\). \(m\) and \(\sigma\) are given as data for each fuzzy term. For temperature domain, \(m\) and \(\sigma\) values are given in Table 2. The numerical domain of temperature is taken from -30 to +45 °C. The central value, \(m\), is a crisp value in the numerical domain at which the membership value to the fuzzy term reaches the highest level. Each fuzzy term in a domain has a different central value. For example, consider the central value of the fuzzy term \(\text{low}\) given in Table 2. The central value of \(\text{low}\) is 0. When we look at the membership graph given in Fig. 3, we see that the graph of the fuzzy term \(\text{low}\) reaches the highest level at 0 and decreases while we are going backward or forward from that point. Since there is no strict border between fuzzy terms, \(\sigma\) values are chosen such a way that they overlap each other as seen in Fig. 3. \(m\) and \(\sigma\) must be appropriate values for each fuzzy term in order to cover all the numerical domain depending on the meaning of fuzzy terms.

Table 2

<table>
<thead>
<tr>
<th>Term</th>
<th>(m)</th>
<th>(\sigma)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frigid</td>
<td>-30</td>
<td>10</td>
</tr>
<tr>
<td>Very-low</td>
<td>-15</td>
<td>10</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Normal</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>Mild</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>Hot</td>
<td>45</td>
<td>10</td>
</tr>
</tbody>
</table>

Fig. 3. The graph of the membership functions of temperature.

Let us take a crisp value and calculate its membership degree to each of the fuzzy terms.

\(\text{istanbul.temperature} = 10, \quad \mu_{\text{frigid}}(10) = 0,\)
\(\mu_{\text{very-low}}(10) = 0.04, \quad \mu_{\text{low}}(10) = 0.61,\)
\(\mu_{\text{normal}}(10) = 0.88, \quad \mu_{\text{mid}}(10) = 0.14,\)
and \(\mu_{\text{hot}}(10) = 0.\)

Now we can say that the temperature of Istanbul is normal with a membership degree 0.88, low with a membership degree 0.61, and so on.

The FOOD model allows both crisp and fuzzy values. While querying the database, if a comparison between a fuzzy and a crisp value is required, the system calculates the membership degree of crisp value to the fuzzy term automatically. If membership degree is greater than or equal to the threshold value given by the user or predefined system threshold value, then the object satisfies the given condition. Thus, this object will be retrieved from the database as an answer to the query. For example, consider the temperature of the city Istanbul given above. If the temperature of Istanbul is 10 in the database and if the user asks the list of cities having “normal” temperature with a membership degree greater than or equal to 0.8 (the threshold value), then the system automatically calculates the membership value of 10 to the fuzzy term “normal”.

Thus, city Istanbul is an answer to the query, since the result is calculated as 0.88 and this value is greater than the given threshold value 0.8.

3.2. Object/class level

The object/class level denotes the membership degree of an object to a class. The main feature that distinguishes the fuzzy classes from crisp classes is that the boundaries of fuzzy classes are imprecise. The imprecision of the attribute values causes
imprecision in the class boundaries. Some objects are full members of a fuzzy class with a membership degree 1, but some objects may be related to this class with a degree between 0 and 1. In this case they may be still considered as instances of this class with the specified degree in [0, 1]. In this model a formal range definition indicating the ideal values for a fuzzy attribute is given in the class definition. However, an object can take any value from the related domain. So, the membership degree of an object to the class is calculated using the similarities between the object values and the class range values, and the relevance of fuzzy attributes. The relevance denotes the weight of the fuzzy attribute in determination of the boundary of a fuzzy class. If an object has the ideal values for each fuzzy attribute then this object is an instance of that class with a membership degree of 1. Otherwise, it is either an instance with a membership degree less than 1, or it is not an instance at all (when the membership degree smaller than the threshold value) depending on the similarities between object values and formal range values. That is, the closer the object value to the range, the higher the membership degree of the object. If the value of an object is crisp, the membership degree of this crisp value to the fuzzy terms in formal range definition is calculated and used to find out the object membership degree to the class. The system calculates the membership degree of objects to their classes while object creating and updating by using the formulas derived by Aksoy and Yazici [3]. The formulas are given briefly in related semantics below. Very detailed discussion and examples about the formulas and semantics can be found in [3,21]. Here we will only give some examples to clarify the concepts.

To calculate the membership degree of an object to a class, we must calculate the inclusion degree of attribute values with respect to the range of attributes. Since the attribute values may be connected through AND, OR or XOR semantics, the inclusion value is dependent on the attribute semantics. The formulas used to calculate inclusion degrees are briefly given below for each connection semantics. The inclusion degree is denoted by “INC” in the formulas.

If \( a_i = \emptyset \), then \( INC = 0 \) for all semantics, where \( o_j \) is an object and \( a_i \) is an attribute of object \( o_j \); otherwise

Case 1. AND semantics: AND semantics requires that all of the instances appear at the same time, no other object meanwhile is present. If an object has all of its values in the range, the inclusion degree is one. Otherwise, it is less than one depending on the similarities. The formula for AND semantics is

\[
INC(rng(a_i)/o_j(a_i)) = \text{Min}\left[\text{Min}\left[\text{Max}(\mu(x,y))\right], \text{Min}\left[\text{Max}(\mu(z,w))\right]\right],
\]

where \( x \in rng(a_i), \forall y \in o_j(a_i), z \in o_j(a_i), \forall w \in rng(a_i) \).

For an example, let us consider the range definition and object values given below and calculate the inclusions. The similarity relation of the attribute contaminant is given in Table 3.

\[
range(\text{contaminant}) = \langle \text{strontium, iodine} \rangle,
\]
\[
o_1, \text{contaminant} = \langle \text{strontium, iodine} \rangle,
\]
\[
o_2, \text{contaminant} = \langle \text{iodine, zinc} \rangle,
\]
\[
o_3, \text{contaminant} = \langle \text{iodine} \rangle,
\]
\[
INC(\text{rngPOLLUTANTS}(\text{contaminant})/o_1(\text{contaminant}))
\]
\[
= \text{Min}\left[\text{Min}\left[\text{Max}(\mu(\text{strontium}, \text{strontium}), \mu(\text{iodine, iodine})), \text{Max}(\mu(\text{iodine, strontium}), \mu(\text{iodine, iodine})))\right], \text{Min}\left[\text{Max}(\mu(\text{strontium}, \text{strontium}), \mu(\text{iodine, strontium})), \text{Max}(\mu(\text{iodine, strontium}), \mu(\text{iodine, iodine})))\right]\right]
\]
\[
= \text{Min}\left[\text{Min}\left[\text{Max}(1.0, 0.8), \text{Max}(0.8, 1)\right], \text{Min}\left[\text{Max}(1.0, 0.8), \text{Max}(0.8, 1.0)\right]\right] = 1
\]

<table>
<thead>
<tr>
<th>Contaminant</th>
<th>Strontium</th>
<th>Iodine</th>
<th>Cesium</th>
<th>Lead</th>
<th>Zinc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strontium</td>
<td>1.0</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>Iodine</td>
<td>0.8</td>
<td>1.0</td>
<td>0.9</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>Cesium</td>
<td>0.8</td>
<td>0.9</td>
<td>1.0</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>Lead</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>1.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Zinc</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>1.0</td>
</tr>
</tbody>
</table>
By using the same formulation, the inclusion values of the other objects are calculated as given below:

\[ \text{INC}(\text{rngPOLLUTANTS}(\text{contaminant})/\ o_1(\text{contaminant})) = 0.7, \]
\[ \text{INC}(\text{rngPOLLUTANTS}(\text{contaminant})/\ o_3(\text{contaminant})) = 0.8. \]

We should note that in the GBP model [11], if an object has a subset of the range, its inclusion degree is always calculated as one. Whereas, the ideal value for AND semantics should consist of all the elements in the range not a subset of it. That is, the inclusion degree of an object which has all the elements in the range and an object which has only some elements of the ranges must be different. This is much more realistic when we consider the semantics of the AND operator. A book that can be described as *dark-blue* and *white* is much different than a book defined as *dark-blue* only. In our model we derived a new formulation as given above to handle AND semantics correctly in all possible instances.

**Case 2. OR semantics:** OR semantics uses a subset of the range definition. The similarities among the object values affect the inclusion degree. If similarities among the object values decrease, the inclusion degree also decreases. That is, when similarity among the object values increases, the uncertainty decreases. This property forces objects to have close and therefore meaningful attribute values. The formula for OR semantics:

\[ \text{INC} = \text{Min}[\text{Max}(\mu_s(x, z)), \text{Threshold}(\mu_j(a_i))], \]

where \( \forall x \in o_j(a_i), \forall z \in \text{rng}(a_i) \). Threshold value indicates the minimum level of similarity between the elements of object attribute and can be formulated as following:

\[ \text{Threshold}(\mu_j(a_i)) = \text{Min}[\mu_s(x, z)] \]

where \( \forall x, \forall z \in o_j(a_i) \).

Consider the objects given below and calculate inclusions. The similarity relation of the attribute *dose* is given in Table 4.

<table>
<thead>
<tr>
<th>Dose</th>
<th>Very-high</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
<th>Very-low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very-high</td>
<td>1.0</td>
<td>0.9</td>
<td>0.7</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>High</td>
<td>0.9</td>
<td>1.0</td>
<td>0.7</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Medium</td>
<td>0.7</td>
<td>0.7</td>
<td>1.0</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Low</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>Very-low</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.5</td>
<td>1.0</td>
</tr>
</tbody>
</table>

As an additional note here, in the GBP model, the formulas used for OR and XOR semantics give the same result ignoring the semantic differences. It always gives an optimistic result for OR semantics. It does not consider the dissimilarities between object values. In our model, the level of certainty in the information is measured with the use of the threshold value. For example, it is valid to describe a person as {young, very young}, but it is meaningless to say he is old or young, {old, young}. When the elements of the OR set get more dissimilar to each other, the degree of certainty and the value of the information decreases. In the FOOD model the OR and XOR semantics are also differentiated. For example, the *dose* attribute of \( o_2 \) has two values whose similarity is rather small. Although the GBP model ignores this dissimilarity and concludes 1.0 as inclusion degree, our new formulation takes this into account and concludes 0.2. A detailed comparison of our model and the GBP model with various examples is given in [3].

**Case 3. XOR semantics:** XOR semantics forces only one of the entries in the range to be true.
Assuming equal probabilities for the elements of the attribute value, the inclusion degree is formulated as follows:

\[
INC(rng(a_i)/o_j(a_i)) = \text{Avg}[\text{Max}(\mu_x(x, y))],
\]

where \( x \in o_j(a_i), \forall y \in rng(a_i) \). Some examples are given below for XOR semantics. The similarity relation of exposure-time is given in Table 5.

<table>
<thead>
<tr>
<th>Exposure-time</th>
<th>Very-long</th>
<th>Long</th>
<th>Medium</th>
<th>Short</th>
<th>Very-short</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very-long</td>
<td>1.0</td>
<td>0.9</td>
<td>0.8</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>Long</td>
<td>0.9</td>
<td>1.0</td>
<td>0.8</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>Medium</td>
<td>0.8</td>
<td>0.8</td>
<td>1.0</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>Short</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Very-short</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

In our model it is defined as follows:

\[
\mu_C(o_j) = \frac{\sum INC(rng_C(a_i)/o_j(a_i)) \cdot RLV(a_i, C)}{\sum RLV(a_i, C)},
\]

where \( INC(rng_C(a_i)/o_j(a_i)) \) is the inclusion calculated as above by considering the various semantics among attribute values, \( RLV(a_i, C) \) is the relevance of attribute \( a_i \) to the class \( C \) and is given in the class definition.

We use weighted average to calculate the membership degree of objects. In this way, all the attributes affect the membership degree proportionally to their relevance. Let us assume that the following relevance values for the attributes of the pollutants class are given. Then we can calculate the membership degrees of \( o_1, o_2, \) and \( o_3 \) to the class pollutants as following:

\[
\begin{align*}
\text{relevance}(\text{contaminant, POLLUTANTS}) &= 2.5, \\
\text{relevance}(\text{dose, POLLUTANTS}) &= 1.5, \\
\text{relevance}(\text{exposure-time, POLLUTANTS}) &= 1.0,
\end{align*}
\]

\[
\begin{align*}
\mu_{\text{POLLUTANTS}}(o_1) &= (INC(rng_{\text{POLLUTANTS}}(\text{contaminant})/o_1(\text{contaminant})) \\
& \ast RLV(\text{contaminant, POLLUTANTS}) \\
& + INC(rng_{\text{POLLUTANTS}}(\text{dose})/o_1(\text{dose})) \\
& \ast RLV(\text{dose, POLLUTANTS}) \\
& + INC(rng_{\text{POLLUTANTS}}(\text{exposure-time})/o_1(\text{exposure-time})) \\
& \ast RLV(\text{exposure-time, POLLUTANTS}))/(RLV(\text{contaminant, POLLUTANTS}) \\
& + RLV(\text{dose, POLLUTANTS}) \\
& + RLV(\text{exposure-time, POLLUTANTS})) = (1.0 \ast 2.5 + 0.9 \ast 1.5 + 1.0 \ast 1.0)/(2.5 + 1.5 + 1.0) = 0.97, \\
\mu_{\text{POLLUTANTS}}(o_2) &= 0.57, \\
\mu_{\text{POLLUTANTS}}(o_3) &= 0.9.
\end{align*}
\]
3.3. Class/superclass level

The third level is the class which has fuzziness in the class/superclass relationship and in the associations among classes. Usually, the hierarchies do not include fuzziness. Because, we try to construct a meaningful class hierarchy at the beginning of the design. However, it is not always easy to construct a precise hierarchy because of the possible conceptual distance between a class and its subclasses. In such a case, we can use fuzziness at the class/superclass level and query the system with some degree of uncertainty. Therefore, in the fuzzy object-oriented data model we allow fuzziness in the subclass hierarchy. The question of membership between a class and its superclass arises when a query like “To what extent the class belongs to its superclass” is to be answered. The membership of a subclass is calculated using range definitions and relevance of fuzzy attributes, in a similar way to object/class membership degree. The inclusion degrees of the range values of the subclass C in the range values of the Ci are calculated as following:

\[ INC(\text{rng}_C(a_i)/\text{rng}_C(a_i)) = \min[\max(\mu_x, y)] \]

where \( x \in \text{rng}_C(a_i), \forall y \in \text{rng}_C(a_i). \) (7)

In the GBP model, an optimistic view is taken while calculating class/subclass inclusion degrees. The maximum similarity between a class and its subclass is taken as the inclusion degree. We think that the values which are out of the range of the superclass must also be considered as we do in our model.

We use weighted average to calculate the membership degree of a class to its superclass as it is in object/class level. We assume that the following range and relevance values for the class Sites and Forest are given. Now we can calculate the membership value of the class Forest to the superclass Sites.

\[ \mu_{\text{C}}(\text{-1C}) = \frac{\sum INC(\text{rng}_C(a_i)/\text{rng}_C(a_i)) \ast RLV(a_i, C_i)}{\sum RLV(a_i, C_i)} \] (8)

Table 6

<table>
<thead>
<tr>
<th>Location</th>
<th>W</th>
<th>WSW</th>
<th>WNW</th>
<th>SW</th>
<th>SSW</th>
<th>S</th>
<th>SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>1.0</td>
<td>0.8</td>
<td>0.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WSW</td>
<td>0.8</td>
<td>1.0</td>
<td>0.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WNW</td>
<td>0.4</td>
<td>0.4</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>SSW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>0.9</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>0.9</td>
</tr>
<tr>
<td>SSE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.9</td>
<td>0.9</td>
<td>1.0</td>
</tr>
</tbody>
</table>
The comparison of the results of the class/superclass inclusion degrees between the GBP model and the FOOD model is given in Table 8. We think that these new formulations lead to a truer representation of the uncertainty in class/superclass relationship.

In an object-oriented database model, the specification of a class includes the definition of isa relationships, attributes and method implementations. Here a class includes two additional definitions; a formal range definition from a domain for fuzzy attributes and relevance of the attributes to the class, as exemplified in Fig. 4. Since each class within a class hierarchy has a formal range definition for each fuzzy attribute associated with it, one can derive membership values in case of more than one superclass (multiple inheritance). A formal range definition for each class ensures a declared range for a conflicting fuzzy attribute. The fuzzy object-oriented database model (FOOD) is implemented on EXODUS Storage Manager and the details of the implementation is given in [3].

Association among the classes not being in the class/subclass relationship is not directly supported by object-oriented database modeling. That is, there is no constructs like relationship types of EER model [10] to define such associations among classes in any OODB model. In addition, fuzzy associations, some other complex properties, and various associated constraints are also difficult to realize by only utilizing object-oriented database modeling. In this study, we couple fuzzy logic with the FOOD model described above to define such complex and uncertain relationships and associations and associated constraints among classes. In the next section, we show how logic is utilized to overcome some difficulties for modeling such complex and uncertain properties. Before we proceed further it is worth having a discussion on why logic needs to be coupled with object-oriented database modeling for some complex applications existing in the real world.

4. The reasons for coupling object-oriented database modeling with logic

For some applications such as Expert Database Systems, storing and manipulating only data are not adequate. We also need to store and manipulate knowledge for drawing inferences, making decision or just answering the queries. Knowledge is information at a higher level of abstraction which is used to define, control and interpret data in a database. Knowledge is typically generated by experts in some domain of expertise.

Logic has traditionally provided a firm conceptual framework for representing knowledge, since it can deal with the notion of logical consequence. Logic programming languages have made possible to represent knowledge in terms of logical descriptions of facts and rules from which appropriate inferences can be drawn automatically. However, logic languages have some weaknesses for handling complex applications. Some properties of object-oriented database modeling can consolidate logic systems for their weaknesses. These properties may be stated as following:

1. Object-oriented modeling has a powerful abstraction mechanism. Data and operations are encapsulated into objects. Communication among objects and the rest of the system is performed by a set of predefined messages. Data structures are flexible due to the hierarchy from generality to specificity in which data can be stored at the appropriate level of abstraction. Data values can be inherited rather than duplicated. These properties

<table>
<thead>
<tr>
<th>Attribute</th>
<th>INC(GBP)</th>
<th>INC(FOOD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>Temperature</td>
<td>1.0</td>
<td>0.6</td>
</tr>
<tr>
<td>Slope</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Slope</th>
<th>Gradual</th>
<th>Flat</th>
<th>Steep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradual</td>
<td>1.0</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>Flat</td>
<td>0.6</td>
<td>1.0</td>
<td>0.4</td>
</tr>
<tr>
<td>Steep</td>
<td>0.2</td>
<td>0.4</td>
<td>1.0</td>
</tr>
</tbody>
</table>
encourage a structured implementation style and improve the maintainability of programs. Because of that, coupling of object-oriented database modeling with logic provides an easy way for modeling complex applications [6,10,17,25].

2. Modularity of object-oriented modeling results in a stronger modular system when it is used along with logic programming. Modularity refers to the part of a large program which can be developed, tested and modified on its own. In logic, a program consists of a set of clauses. Sometimes the number of clauses may reach several thousands. There is no notation of a hierarchical decomposition such as found in Pascal or C. Due to this flat structure, large programs can be difficult to construct because of their huge size. In an object-oriented system each class is a module with its attributes and methods. Both the behavioral and structural properties of a class are defined in the class definition. This uniform structure eases the design of the complex applications [6,10,23].

3. One of the goals of object-oriented database modeling is to maintain a direct correspondence between real-world and database objects so that objects do not lose their integrity and identity. Identity is the property of an object that distinguishes it from any other object. This is typically represented by a unique object identifier which is independent of any object attributes. Object identity solves many problems of value-based systems such as update anomalies. These features also help the developers and users of applications to have a better understanding of the applications. Thus, objects can be easily identified and operated upon [6,10,37].

The object-oriented database modeling [4,12,37] uses objects to represent entities and organizes the miniworld in the form of class hierarchies. Each object has an identity which is independent of its attribute values. The internal data structure of an object is hidden and the communication is done by sending messages among objects. An object answers the message by selecting the appropriate method definition. Another important feature of this modeling is inheritance. A subclass inherits both attributes and methods from its superclasses without redefining them. With all these and other features of object-oriented data modeling result in a powerful modeling approach for complex applications. However, object-oriented database modeling also has some weaknesses for dealing with some complex applications. Some contributions of logic to object-oriented database modeling for consolidating its weaknesses are as following:

1. Object-oriented database modeling is not very good at specification and manipulation of knowledge. Specification and Deduction are often
necessary for some complex knowledge-intensive applications requiring object-oriented database modeling. Performing deduction with imperative programming makes the programs difficult to debug and maintain. [6, 8]

2. Since there is no commonly accepted query language for the object-oriented database model, another limitation of object-oriented database modeling is querying the database. A query is a single interaction between an end-user and a database. The user formulates a request in a non-procedural manner, specifying what to do rather than how to do it. There are no accepted standardized high level query languages for current object-oriented database models. To solve the query problem, object-oriented database systems can use graphical user interfaces (GUI) and query languages which are extensions of either relational query languages or deductive languages [4]. Graphical user interfaces have some fundamental problems. One of which is that it is not possible to represent and access huge database with possibly millions of objects using icon. The other one is that it is not possible to represent the declarative structure such as a query or a constraint graphically. GUI can not satisfy the query needs of an object-oriented database system on its own. Some contributions of logic to object-oriented database modeling for query requirements are as following:

(a) An object-oriented database model is limited to answering only direct queries. Whereas, knowledge-based systems, using knowledge representation schemes, can answer queries that involve deductions over the stored data [6, 28, 31].

(b) In an object-oriented database model there is a procedure to search a method implementation for answering a message. A search order must be given explicitly. If a method is redefined in a subclass then it overrides all the previous definitions. The message passing mechanism finds the first method in the search order and implements it. It succeeds getting an answer or fails giving an error message. It is not possible to invoke more than one method and sometimes it is not easy to give a search order so that the right method is invoked and implemented. In logic, each method (clause) is a different solution. Sometimes all the possible solution may be asked or if the first method fails then the next method may be implemented instead of failing and stopping evaluation [22, 23].

(c) We can access the data only using the predefined methods because of encapsulation. There are some advantages of encapsulation but predetermining all the operations by a fixed set of methods is a rigid constraint which limits the query capability. In an object-oriented environment, doing a simple retrieval of information can be a complex task if only the message passing mechanism is used. Therefore, utilizing a query language is an important requirement to increase the query capability of the object-oriented database models. Logic, as used in some studies [19, 20], is convenient to be utilized as the query language of the object-oriented database models.

(d) Another requirement of a query language is the set processing capability. A message is to be sent not only to a single object but a collection of objects. For querying single object, the object-oriented database model is adequate. If the query is related to more than one object, some of the object-oriented database systems, like Vbase and Iris, have query languages based on SQL for handling set processing. Another solution is to incorporate a logic language with object-oriented database models [19, 20]. Logic languages are very powerful on set-oriented processing.

3. Today, although the declarative style of logic languages is an important desire, object-oriented systems are heavily dependent on procedural languages [28]. If incorporation is succeeded, the English-like declarative expressions of logic programs can help the application developers and users to understand the model well and modify it easily [6].

4. It is not always possible to specify every relationship in purely object-oriented database modeling easily, since there is no direct way to represent relationships and various associated constraints in this modeling approach. Some applications may even have some relationships, associations, and constraints which include uncertainty; thus, it is even more troublesome to handle such relationships and associations among classes. Association is used to associate objects from several independent classes. It involves the possibility of incorporating objects related by a particular relationship into a new integrated object.
Now, consider the Environmental Information System introduced in Fig. 1. The pollution forms when a pollutant and a site are located in the same place. Pollution is a relationship between a pollutant and a site. It includes both the properties of these two classes and these two classes are independent of each other. Therefore, we define an association between these two groups of classes to obtain a pollution class whose instances are associations of instances of some of the pollutant classes (Gaseous, Liquids, and/or Solids) and some of the site classes (Forest, Agri-Area, WaterSupply, and/or City). Since there is no direct way to represent associations in object-oriented database modeling, the pollution association is represented by inheritance in Fig. 1. The pollution class inherits from the pollutants classes and the sites classes. This lattice seems meaningful to represent the pollution. However, this lattice forms a categorization which is hard to handle with object-oriented database modeling alone. A category is a subclass of multiple classes whose properties may be different and the category class might be inheriting its properties from a subset of these superclasses. In Fig. 1, the pollution class is an example of a category. For example, a pollution object which represents the air pollution in a city, inherits its properties from the gaseous class and the city class. The Environmental Information System depicted in Fig. 1 is a relatively simple class lattice but still hard to handle with object-oriented modeling features alone.

Furthermore, there are some factors to be considered for pollution. For example, there are many contaminants in each subclass of pollutants, the effect of each contaminant to each site is different, the effectual amount of each contaminant to each site is different, and so on. The pollution relationship includes many rules. The pollution class inherits from multiclasses and becomes a complex one to handle.

Instead of combining all pollution associations in one class, i.e., pollution in Fig. 1, constructing a class lattice as in Fig. 5, forms simpler classes to handle. Although the classes become simpler, many new classes appear and the lattice becomes very complex. Therefore, it is still difficult to resolve some of the problems such as handling many virtually created new classes to represent various associations among existing classes. Each association among classes is represented by a new class. In a system, many associations may be required to satisfy different query requirements from different users, which will result in inefficiency, since it is not easy to construct a new class for every possible association. To define a new association or to update an existing association, the schema definition of the system represented by a lattice must be updated. Although there exist some proposed solutions, the schema modification problem in object-oriented database models is still a research issue [10, 37].

Since it is not always easy to handle associations in purely object-oriented database environment,
our solution for such problems is to keep the object-oriented hierarchy simple but incorporate logic in our modeling approach. Thus, it becomes easier to handle some of the associations, complex and uncertain relationships and associated constraints. As the solution to the example problem, we apply our modeling approach to the Environmental Information System as in Fig. 6.

Along with this class hierarchy, we use logic for handling associations as follows: The membership degrees of objects to their classes are calculated as explained in Section 3. If there is an object of the class pollutants and an object of the class sites in the same location, then pollution occurs with some degree. This association is defined utilizing IF-THEN rules. For example, consider the association between the class gaseous and the class city which represents air-pollution objects. Assume the following class definitions:

```
CLASS gaseous
  PROPERTIES
  dose
  exposure-time
  location
  END

CLASS city
  PROPERTIES
  location
  END
```

This association is defined simply as following:

**R-1:** IF a gaseous-pollutant with certain dose and exposure-time is located over a city THEN air-pollution occurs in this city.

In this rule, to check whether the gaseous pollutant is located over a city, the location attribute of gaseous-pollutant and the location attribute of city are compared. The attributes dose and exposure-time of gaseous-pollutant are also checked, since these values must be of a certain degree for an occurrence of pollution. In our modeling approach, the associations are established utilizing attributes of the existing classes of the hierarchy and they are represented by a new class, i.e. air-pollution. The associated objects satisfying specified conditions are determined by utilizing deductive capability of logic. If association rule R-1 succeeds for two objects, one is an instance of the class gaseous and the other is an instance of the class city, then an air-pollution object is formed from these two objects and the corresponding classes are associated with each other.

If we did not use logic to define associations, the association established by rule R-1 would be defined by class PK in the class lattice given in Fig. 5 or would be defined in the class pollution in the lattice given in Fig. 1. In this case, as mentioned above, many possible problems would be occurring. Therefore, we chose to utilize logic to overcome those problems. In this way, various association rules which associate existing classes in a class hierarchy can be defined. Furthermore, new associations can be defined utilizing previously defined associations, that is, nested associations can also be defined. Consider a nested association illustrated in Fig. 7 as an example. The association air-pollution is already defined by the rule R-1. We define the following association between the class liquids and the class water-supply.

```
Fig. 6. The simplified class hierarchy of the environmental information system.
```
R-2: *IF there is liquids-pollutant with certain dose in a water-supply AND this water-supply is used for drinking in a city THEN water-pollution occurs.*

Now, we can define the new association *emergency-pollution* utilizing previously defined associations *air-pollution* and *water-pollution* with the following rule:

R-3: *IF air-pollution occurs in a city AND water-pollution occurs in the same city THEN emergency-pollution occurs in this city.*

R-3 is successful if R-1 and R-2 are successful. That is, if both the air of the city and water supply of the city are polluted, then *emergency-pollution* occurs in this city. Thus, in our approach, we can define arbitrary level of nested associations very easily by incorporating logic in our modeling.

From all these, we conclude with our proposition that instead of using the class hierarchy of the Environmental Information System represented in Fig. 1 and another equivalent one shown in Fig. 5, the finalized simplified hierarchy, shown in Fig. 6, along with the logical rules and facts should be used to handle the overall system more easily and accurately. In our modeling approach, the association issue is resolved by coupling logic into the object-oriented database modeling. In this way, new class requirements to define associations among classes are eliminated and the class hierarchy is kept compact and therefore simpler. New associations can be defined easily by inserting new rules, existing associations can be changed easily by deleting or modifying related rules without affecting the schema definition represented with the simplified hierarchy. The ability to define nested association is an important capability of our modeling approach. The claims made here is also justified by the implementation of the system and explained in Section 6. In the implementation, it is possible to modify the defined associations even when the system is running. Whereas a modification in class schema requires recompilation of schema definition.

Because the associations among classes may also involve in some types of uncertainty, the logic to be utilized in our model must also handle the uncertainty issues. To satisfy this requirement, fuzzy logic [35, 36] is incorporated into our modeling approach, which is described in the next section.
5. Incorporating fuzzy logic for handling uncertainty

Knowledge within the decision support tool is represented using the *IF-THEN* rules as follows:

**IF antecedent THEN consequent.**

In fuzzy logic, the same format is used. The input–output relation of the system is expressed utilizing *IF-THEN* rules in which the antecedent and consequent involve linguistic variables [15, 27, 35]. For example,

**R-4:** *IF dose is high*  
*THEN pollution is very-dangerous,*

where dose and pollution are linguistic variables, *high* and *very-dangerous* are fuzzy values. The antecedent of a rule may be composite of more than one clause connected by the fuzzy logical operators *AND* and *OR*. For example,

**R-5:** *IF dose is high AND exposure-time is long*  
*THEN pollution is very-dangerous,*

The rules can be used to define various relationships and associated constraints among not only crisp but also fuzzy attributes of different classes; therefore, associations among relating classes along with their fuzzy properties can be defined with our model naturally and easily. These rules usually are extracted from the experts of the field. Variables of rules represent attributes of classes. For example,

Domain(dose) = \{very-high, high, medium, low\},  
Domain(exposure-time)  
= \{very-long, long, medium, short\},  
Domain(pollution)  
= \{extremely-dangerous, very-dangerous, dangerous, less-dangerous\}.

The experts will define the rules using all these fuzzy values by taking into account all possible combinations of the linguistic values as follows:

**IF gaseous.dose is very-high**  
**AND gaseous.exposure-time is very-long**  
**THEN pollution.level is extremely-dangerous.**

...  
**IF gaseous.dose is low**  
**AND gaseous.exposure-time is short**  
**THEN pollution.level is less-dangerous.**

Depending on the object attribute values, one of the rules is activated and produces a conclusion. In our model, since there is no strict border among fuzzy values we use similarity relations to define the similarities among the elements of the fuzzy domain. Because of that, there is no need to define so many rules including all the combinations of fuzzy terms. We only define some main rules. If there is no exact matching, the similar rules are activated. A related example is going to be given later. In a fuzzy logic system, any rule antecedent with a non-zero membership value causes that rule to be activated. In order to eliminate undesired effects, a threshold value (a priori specified value) is employed. The membership grade of consequent is obtained from those of antecedent. For example, consider the rule below:

**R-6:** *If gaseous.dose is very-high AND exposure-time is low*  
*THEN pollution.level is extremely-dangerous.*

If the fact shows that the level of *dose* is *very-high* and the level of *exposure-time* is *very-long*, this rule is activated. However, if the values of object do not match exactly with the rules, there may be a partial matching. Since there is no strict border between fuzzy terms in fuzzy logic, partial matching arises [15, 36]. Here, partial matching is used to mean that not only the rule with exact matching is activated but also any other rule is activated even if the value of object is not exactly equal to the value of that rule but similar to. For example, even if the level of *dose* is *high* instead of *very-high*, the rule R-6 will be activated if the similarity of *high* to *very-high* is greater than the priori given threshold value.

Since the system allows both fuzzy and crisp values, some objects may have fuzzy values and some may have crisp values for the same attribute. If the attribute value is fuzzy, the rules are activated using similarity relations as explained above. If the attribute value is crisp, then the membership degree of this crisp value to the fuzzy term in the rule is
calculated. The calculation of crisp values to fuzzy terms was already discussed in Section 3. If membership degree is greater than or equal to the threshold value, the rule gets activated.

If there is an exact matching between the fact and the rule, the membership value of consequent will be one. If there is a partial matching then the membership degree of consequent is calculated using the similarities of the antecedent clauses. Since the effects of each variable to the conclusion may be different, a relevance value is determined for each variable. Utilizing these relevance and membership values of antecedent, the membership value of consequent is calculated. In this system, we use the relevance of attribute in class definition for the relevance of rule antecedent clauses. The formulas used to calculate the membership degree of conclusion are given below depending on the logical operators of antecedent clauses.

for **AND** operator:

\[ \mu_i = \min\left(\frac{\mu_1 \cdot r_1 \cdot \mu_2 \cdot r_2 \cdot \ldots \cdot \mu_n \cdot r_n}{r_{\text{max}}, r_{\text{max}}, \ldots, r_{\text{max}}} \right) \]

(9)

for **OR** operator:

\[ \mu_i = \max\left(\frac{\mu_1 \cdot r_1 \cdot \mu_2 \cdot r_2 \cdot \ldots \cdot \mu_n \cdot r_n}{r_{\text{max}}, r_{\text{max}}, \ldots, r_{\text{max}}} \right) \]

(10)

where, \(1 \leq j \leq n\), \((\mu_1, \mu_2, \ldots, \mu_n)\) are the membership degrees of compared objects, \((r_1, r_2, \ldots, r_n)\) are the relevance values of attributes in their classes.

For example, let us consider the following rule and calculate the membership degree of consequent for the values given below:

**R-7:** IF liquids.dose is very-high OR liquids.dose is high
AND liquids.exposure-time is very-long
AND liquids.location is near by a forest.location
THEN pollution.level is very-dangerous for that forest.

For example, let us assume the values for objects, similarities, relevance and threshold are given as following:

\[ \text{pl.dose} = \{\text{high}\}, \quad \text{pl.exposure-time} = \{\text{long}\}, \]
\[ \text{pl.location} = \{\text{sw}\}, \quad \text{fl.location} = \{\text{sw}\}, \]
\[ \mu_s(\text{very-high, high}) = 0.8, \quad \mu_s(\text{very-long, long}) = 0.9, \]
\[ \text{relevance}(\text{dose, POLLUTANTS}) = 2.5, \]
\[ \text{relevance}(\text{exposure-time, POLLUTANTS}) = 1.5, \]
\[ \text{relevance}(\text{location, POLLUTANTS}) = 1.0, \]
\[ \text{relevance}(\text{location, FOREST}) = 2.0, \] threshold = 0.3.

For these values, R-7 is activated and produces a conclusion. The membership degree of conclusion is calculated as follows:

\[ \mu_{\text{very-dangerous}} = \min(\max(\mu_s(\text{high, very-high}) \cdot r_s(\text{dose, POLLUTANTS}) \cdot r_s(\text{location, POLLUTANTS}) \cdot r_{\text{max}}), \]
\[ \mu_s(\text{long, very-long}) \cdot r_s(\text{exposure-time, POLLUTANTS}) \cdot r_{\text{max}}, \]
\[ \text{avg}(\mu_s(\text{sw, sw}) \cdot r_s(\text{location, POLLUTANTS}) \cdot r_{\text{max}}), \]
\[ \mu_s(\text{sw, sw}) \cdot r_s(\text{location, FOREST}) \cdot r_{\text{max}})) \]
\[ = \min(\max(0.8 \cdot 2.5/2.5, 1 \cdot 2.5/2.5, 0.9 \cdot 1.5/2.5, \text{avg}(1 \cdot 1/2.5, 1 \cdot 2/2.5))), \]
\[ = \min(1, 0.54, 0.6) = 0.54. \]

The pollutant pl is very-dangerous for the forest fl with a membership degree 0.54.

When more than one rule are activated, depending on the user’s need, either all of the conclusions
having membership degree above the given threshold value are considered or the conclusion with the greatest membership degree is considered as the best solution. Since an object is allowed to take a set of fuzzy attributes, sometimes we can come across objects which have more than one value for a fuzzy attribute. In such a case, the fuzzy value which has the greatest similarity degree to the rule is considered.

Now, utilizing the information given above, we can define more complete fuzzy rules which represent various uncertain pollution associations between the pollutants and the sites classes as examples:

R-8: IF gaseous.contaminant is SO$_2$
AND gaseous.dose is very-high
AND gaseous.exposure-time is very-long
AND gaseous.location is near by a city.location
THEN airpollution.level is extremely-dangerous in this city for people.

Here, a fuzzy association between the class gaseous and the class city is defined. As seen in the rule, variables of the rule represent attributes of classes, i.e. gaseous.contaminant is the contaminant attribute of the gaseous class. Airpollution is the name of the new class representing the defined association. If rule R-8 succeeds for any two objects, one from the class gaseous and one from the class city, then an airpollution object is formed with the additional attribute “level” giving the level of pollution as a fuzzy value, extremely-dangerous. So, an airpollution object associates a gaseous object and a city object with a specific role represented with an additional attribute, level.

An association class only represents a relationship among classes. Now, we can go one step further and create a new class deriving its attributes from existing ones in addition to associations. The created class may inherit all the attributes or may selectively inherit some attributes from related classes. Furthermore, some attributes can be derived from existing properties. The instances of such a derived class can be stored if desired. In this way, users can define new classes from their own point of view without changing the schema of the system. The rule R-8 is modified to create a new class named city-pol as shown with the following rule:

R-9: IF gaseous.contaminant is SO$_2$
AND gaseous.dose is very-high
AND gaseous.exposure-time is very-long
AND gaseous.location is near by a city.location
THEN create the class city-pol with attributes city-pol.contaminant
= gaseous.contaminant, city-pol.dose
= gaseous.dose, ..., city-pol.location
= city.location.

6. Implementation

6.1. POPLOG environment

POPLOG is an integrated, interactive, multi-language software development environment. Its core language is POP-11. Besides POP-11, it contains incremental compilers for Prolog, Lisp and ML. All of these compilers generate code from the same virtual machine (VM) instruction set. The VM code is translated into machine code. POPLOG also provides a number of facilities for loading and running external code which has been compiled by a non-POPLOG compiler, for example C, Pascal or Fortran.

Pop-11 is a powerful general-purpose programming language. It is a contemporary of Lisp and shares much of its functionality but has a procedural syntax. For some tasks, such as programs which do a great deal of arithmetic, string or vector manipulation, or deterministic list processing, it is more efficient than Prolog. Like Pascal, it encourages well-structured, clear programs made of independently testable parts. Like Basic, Lisp and Prolog, it is fully interactive, allowing full use of the computer to simplify and accelerate development and testing of programs. POP-11 has a package called Flavours for object oriented programming. The ideas of this package are loosely based on those of the Zetalisp Flavors system but much has been inherited from other object oriented programming systems such as Smalltalk-80, Loops and Common Loops and Simula.
6.2. The structure of the system

The overview of the system is given in Fig. 8. We used Pop-11 Database as the database of the system. We constructed a knowledge-base on Prolog to define the pollution relationships between the pollutants and sites classes. We implemented fuzzy properties such as similarity matrices, domain definitions, utilizing Pop-11. Object-oriented features are realized on Flavours. X WINDOWS is used for user interface with OPEN LOOK widget set.

The structure of the system is seen in Fig. 9. There are three main operations on the system. These are database management, rule management and queries. Database management includes adding new objects into database, updating and deleting existing objects utilizing object-oriented features. Rule management includes adding new rules, updating and deleting of existing rules.

The queries consist of two main group of queries. In the first group, there are some predefined queries which are used very frequently. These queries are accessed by menu without writing anything. There are also a fuzzy query processor and an object browser in this group. The second group provides an interactive mode usage of the system selecting one of the Pop-11, Flavours and Prolog options. Users can form any query utilizing Poplog features. Some of the important issues related to the implementation are summarized in the coming subsections.
6.3. Class definition

Classes are defined utilizing Flavours. When an object is created, it is inserted automatically into the database utilizing initialize method. A brief class definition is given below as example:

```
flavour Sites;
  ivars name;
  ..........
  defmethod location-> result;
    if present(['name == [' == location
     ? result == ]]) then return;
    else false-> result;endif;
  enddefmethod;
  defmethod updaterof location(temp);
    if present(['^name ??c [ == location
    ?n ??r])
      then
        remove(it);
      endif;
      add(['name ~'c [location ^temp ^^r]);
  enddefmethod;
  ..........
endflavour;
```

For each attribute, an access method and an updater method are defined. The access method defines how to get the attribute values, the updater method defines how to update the attribute values in the database. Objects are stored in the following format in the database:

```
[id class membership-value [attribute-1 value-1
  attribute-2 value-2 ... attribute-n value-n]]
```

6.4. Definition of fuzzy features

The ranges and the relevance values are defined as methods in each class definition. Range and relevance of each attribute can be accessed by sending messages. For example, to get the range of attribute temperature, the following message is sent.

```
object-id <-range("temperature");
```

A function is defined to facilitate accessing of desired similarity. For example, to find the similarity of high to veryHigh temperature, the following function call is used.

```
similarity("temperature", "high", "veryHigh");
```

The first parameter is the domain name, the second and third parameters are the fuzzy terms whose similarity is required.

For the relationship between fuzzy and crisp values, the formula (1) is used. The membership degree of a crisp value to a fuzzy one is calculated and returned by the function fuzzify. Its format is as following:

```
fuzzify(attribute, crisp-value, fuzzy-value);
```

6.5. Database management

New instances are created utilizing system procedure make_instance. This procedure takes a list as parameter. The first item in the list is the class name, the other items consist of attribute and value pairs as seen below:

```
make_instance([class attribute-1 value-1
  attribute-2 value-2 ... attribute-n value-n]);
```

In this procedure, the message new is sent to the given class and so, a new object is created. Then, the message initialize is sent to the new object. In the initialize method, the membership value of the object to the class is calculated and new object is inserted into database in the defined format.

Updating an object value is done utilizing updater method of the related attribute. Updating format is as follows:

```
new-value-> object-id <-attribute;
```

The updater method takes the new value as parameter and changes the old value with the new one. The object membership value is recalculated automatically after updating.

Deletion of instance is done by means of destroy procedure. The id of the instance to delete is given as parameter as following:

```
destroy(object-id);
```
6.6. Knowledge-base

The knowledge-base is constructed on Prolog. It includes various pollution rules and some utility rules such as database access rules. It is possible to modify the knowledge-base. So, new rules related to pollution can be inserted into the knowledge-base or existing rules can be updated or deleted. A rule which takes a gaseous pollutant and a city and checks whether the pollutant creates a severe air pollution is defined as following on Prolog:

\[
\text{airpollution}(P, F, \text{severe}, MV) :\text{-}
\begin{align*}
\text{classname}(F, N) &\text{, } N = \text{'city'}, \\
\text{contaminant}(P, C1) &\text{, } dose(P, A1), \\
\text{airpollution}(\text{dose}, \text{severe}, A2), \\
\text{exposuretime}(P, B1), \\
\text{airpollution}(\text{exposuretime}, \text{severe}, B2), \\
\text{similar}(\text{exposureTime}, B1, B2, \text{false}, MV2), \\
\text{airpollution}(\text{contaminants}, N, C2), \\
\text{similar}(\text{contaminant}, C1, C2, MV3), \\
\text{location}(P, D1), \\
\text{location}(F, D2), \\
\text{similar}(\text{location}, D1, D2, MV4), \\
\text{combine}([MV1, MV2, MV3, MV4], MV), \\
\text{threshold}(\text{mv}, T), MV > T.
\end{align*}
\]

6.7. Coupling of database and knowledge-base

Coupling of database and knowledge-base is an important issue in the implementation. The knowledge-base has to access to the database to get the objects. On the other hand, we need to access to the knowledge-base from Pop-11 for queries. We used Poplog utilities for these requirements. The procedure \text{prolog\_invoke} is used to access to the knowledge-base from Pop-11. This procedure fires a given Prolog goal and returns the result of the first successful rule. A program piece showing the usage of this procedure is given below.

... 
\text{prolog\_newvar( )} \Rightarrow X; \\
\text{prolog\_newvar( )} \Rightarrow Y; \\
\text{prolog\_newvar( )} \Rightarrow Z; \\
\text{prolog\_maketerm} (p, c, X, Y, \text{‘airpollution’}, 4) \Rightarrow g; \\
\text{if prolog\_invoke(prolog\_maketerm([A X A y], g, Z, \text{‘findall’}, 3)) then} \\
\text{prolog\_full\_deref(Z)} \Rightarrow z; \\
\text{for i in z do} \\
\text{[The air pollution \’p is \’q for city \’c with mv \’z] = >} \\
\text{endfor; \\
endif; \\
...}

To access to Database from Prolog, the system predicate \text{prolog\_eval} is used. This predicate takes a Pop-11 function name with its parameters, executes it and returns the result. A rule to access to the \text{location} attribute of an object is defined as following:

\[
\text{location}(X, Y) :\text{-}
\begin{align*}
\text{location}(X, Y) :\text{-}
\end{align*}
\]

Convert is a function defined in Pop-11. It takes an object id and a message as parameter, sends this
message to the given object, returns the result of the method implemented. The result of the function `convert` is assigned to the variable Y. In this way, it is possible to access to any attribute of any object. For example,

```
?- location(p, X).
X = [w]
```

Another database requirement is the need of getting a set of objects from database. Two predicates are defined to satisfy this requirement. The first one, `objects`, gets the objects which satisfy the given conditions regardless of fuzziness and it is defined for efficiency. The second, `fuzzyobjects`, gets the objects which satisfy the given conditions regarding fuzzy features and can handle both fuzzy and crisp conditions. For the predicate `objects`, a query example which gets `waterSupply` objects whose usage is `drinking` and kind is `dam` is given as

```
objects(waterSupply, [[usage = drinking] [kind = dam]], X).
```

The first parameter is the class name and the second parameter consists of attribute name and value pairs. An example, which gets some of the objects of `forest` class with a condition, is given below for the predicate `fuzzyobjects`:

```
fuzzyobjects(forest, 0.5, [[temperature >= 0] and [precipitation = [heavy]]], [temperature = 0.8 precipitation = 0.8], F).
```

The first parameter is the class name (i.e. `forest`), the second is the level of object membership value (i.e. 0.5), third is the condition (i.e. `[[temperature >= 0] and [precipitation = [heavy]]]`) and the fourth is the level of attributes (i.e. `[[temperature = 0.8 precipitation = 0.8]]`). The result is returned in `F`. The attribute levels are given as 1 for crisp attributes.

Using these features, various rules are defined on the knowledge-base. The knowledge-base gets the required objects from the database and applies the rules on them.

6.8. Queries

Some of the queries which can be required very frequently are predefined for easy access. These queries can be accessed easily by menus utilizing mouse clicks. For example, the list of cities which have air pollution is one of these predefined queries. In addition to predefined queries, user can direct any query to the system in interactive mode, selecting either Pop-11 or Prolog option, as explained below.

Queries can be asked to the system on both Pop-11 and Prolog. On Pop-11, Pop-11 database queries and Flavours queries can be formed. The connection between Flavours and Database are provided in class definitions. Flavours queries are realized by sending messages to the objects. For example,

```
p <- location;
```

where `p` is an object and `location` is a message. For recursive queries, Pop-11 database facilities can be used. For example, a query to list all objects of the class `pollutants` can be formulated as following:

```
foreach([ = pollutants == ] do it =) endforeach;
```

The most important queries on the system are the ones which use the knowledge-base on the Prolog. On Pop-11, we can form prolog goals and fire them utilizing various ways. For example,

```
: ?- pollution(pl, fl, X).
```

Here, “?-” is a macro which activates the given prolog goal as if we were on prolog. This method is a simple one which can be used on interactive mode. The procedure `prolog_invoke` supports a more powerful access method as explained in Section 6.7. Furthermore, passing from Pop-11 to Prolog or vice versa is very simple on Poplog. On Pop-11, user calls prolog interpreter writing only “prolog”, activates any Prolog goal and returns back to Pop-11 writing “pop11”. User can call the rules in the knowledge-base utilizing one of these methods.

In the knowledge-base, various pollution rules are defined. Each pollution rule is defined in such a way that different queries are satisfied. For example, let us elaborate `air pollution` rule for different query options.

```
?- airpollution(pl, cl, X).
X = [severe = 0.7]
```
This goal asks whether the *gaseous pollutant* \( pl \) creates pollution for the *city* \( cl \). If rule succeeds, a fuzzy result accompanied by a membership degree is returned. More than one result can be returned depending on object values. In such a case, the result which has the greatest membership value is considered as the best result. In this goal, the results whose membership degree are greater than the default threshold value are returned. The threshold value is 0.5 for all rules. Sometimes, user may want to give a different level. The user can give a different level as follows:

\[
?-\text{airpollution}(pl, cl, 0.8, X).
\]

We can ask whether there is air pollution in the city \( cl \) as follows:

\[
?-\text{airpollution}(X, cl, Y)). \text{ or } \text{airpollution}(X, cl, 0.8, Y).
\]

We can ask whether the *gaseous pollutant* \( pl \) creates air pollution in any city as follows:

\[
?-\text{airpollution}(pl, X, Y)). \text{ or } \text{airpollution}(pl, X, 0.7, Y).
\]

To list all the cities where there is air pollution can be asked as follows:

\[
?-\text{airpollution}(X). \text{ or } \text{airpollution}(X, 0.8).
\]

The format exemplified with airpollution rule is applied to all pollution rule in a similar way.

### 6.9. Fuzzy queries

An SQL like format is used for fuzzy database queries as follows:

```
SELECT attribute-list FROM class-name
OBJECT LEVEL class-membership-level
WHERE conditions
LEVEL attribute-membership-level
```

Fuzzy queries are evaluated and answered by a fuzzy query processor. Both fuzzy and crisp values can be used in the conditions. The comparison operators used are =, >, <, >=, <=. The connection operators AND and OR can be used to connect different conditions. If the class membership value and attribute membership values are not given, default values are considered. Default threshold value is taken 1.0 for both the class membership value and the attribute membership value. That is, if levels are not given or given as 1.0, that query is a crisp one. If there is no condition, all the instance of the given class are listed. A query example is given below:

```
SELECT location humidity FROM forest
OBJECT LEVEL 0.7
WHERE (temperature > 0 and humidity = [medium high]) or precipitation >= [normal]
LEVEL temperature 0.7, humidity 0.8
```

As the result of this fuzzy query, the attributes *location* and *humidity* of the objects of the class *forest*, which have object membership values greater than or equal to 0.7 and satisfy the given condition, are displayed. In the condition, the object having similarities greater than or equal to 0.7 for temperature, 0.8 for humidity and 1.0 for precipitation are retrieved.

### 6.10. User interface

User interface is implemented utilizing OPEN LOOK widget set. A menu-driven system is created. Utilizing the capability of X WINDOWS programming environment, a user-friendly database management and query interface is implemented. Controlling data to enter, correct fuzzy and crisp values are provided to the database. Multiple screen can be opened. The results of queries are returned in answer windows. These result can be copied elsewhere. For interactive mode, a special window is opened, and the session can be saved in a file. The details about the user interface and the other implementation issues are given in [21].

### 7. Conclusions

In this study we proposed a fuzzy object-oriented database modeling approach which incorporates logic for representation of knowledge and for deduction capability to satisfy various users’ needs. First, we introduced a fuzzy object-oriented
database model (FOOD) based on similarity relations to handle fuzziness at three different levels: attribute level, object/class level, class/superclass level. Second, we coupled fuzzy logic with this object-oriented database model to define various complex and uncertain relationships, associations and associated constraints which could not be handled easily with the object-oriented modeling alone. Third, we have used our approach for modeling a complex application, the Environmental Information System, containing uncertainties along with complex objects in its nature. We used fuzzy set theory to deal with uncertainty in this study. Last, for the implementation of our modeling approach, we used POPLOG environment. With the implemented system, object-oriented database modeling with fuzzy logic for utilizing powerful features of each paradigm in a single system is realized. A fuzzy object-oriented database model fully integrated with fuzzy logic for deductive capability could be an important research topic for further research. In addition, research on efficient index techniques and on fuzzy optimization techniques for fuzzy object-oriented databases may be other possible research topics in this area.

References


