IFOOD: An Intelligent Fuzzy Object-Oriented Database Architecture

Murat Koyuncu and Adnan Yazici, *Senior Member*, *IEEE*

Abstract—Next generation information system applications require powerful and intelligent information management that necessitates an efficient interaction between database and knowledge base technologies. It is also important for these applications to incorporate uncertainty in data objects, in integrity constraints, and/or in application. In this study, we propose an intelligent object-oriented database architecture, IFOOD, which permits the flexible modeling and querying of complex data and knowledge including uncertainty with powerful retrieval capability.

Index Terms—Object-oriented databases, knowledge-base systems, uncertainty, fuzzy set theory, flexible querying.

1 Introduction

In recent years, in order to satisfy the requirements of the nontraditional database applications including CAD/CAM, office automation systems, engineering designs, and spatiotemporal databases, etc., new database models, such as the object-oriented database model and the deductive database model, have been developed. The object-oriented database model, which provides powerful data modeling features, has gained more popularity in recent years [2], [9], [13]. The deductive database model is another approach used to achieve intelligent behavior, an important requirement of the knowledge-intensive applications [6], [7], [15]. While these two database models are being developed separately, the interaction and/or integration of database and knowledge base technologies is also an important requirement toward the development of the next generation information systems. This is reflected in the continuing researches toward the development of deductive object-oriented database models since the late 1980s [3], [16], [22], [24], [25], [26], [33].

However, most of the existing database models are designed under the assumptions that the data/information stored is precise and queries are crisp. In fact, these assumptions are often not valid for many of the next generation information systems since they may involve complex information with uncertainty. In general, data/information in databases may be uncertain for the following reasons:

1. A decision in many knowledge-intensive applications usually involve various forms of uncertainty.
2. Integrating data from various sources is not usually a crisp process, while unifying various heterogeneous data into an integrated form, due to possible semantic differences (and other reasons), sometimes forcing data to be completely crisp may result in falsity and useless information.
3. Information in some nontraditional applications is inherently both complex and uncertain, i.e., representing subjective opinions and judgments concerning medical diagnosis, economic forecasting, or personal evaluation.
4. In natural languages, numerous linguistic terms with modifiers (e.g., "very," "more or less," etc.) and quantifiers (e.g., "many," "few," "most," etc.) are used when conveying vague information.

Handling uncertainty in databases was first proposed on relational-based database models [4], [30], [31], [32], [41]. For example, Takahashi [31] discusses the theoretical foundation of query languages to fuzzy databases. He proposes two fuzzy database query languages: a fuzzy calculus query language and a fuzzy algebra query language. Bosc and Pivert [4] extend the capabilities of standard SQL in order to allow the formulation of imprecise criteria addressed to a relational database. A query language, called SQLf, was developed using the fuzzy set theory. Selection, join, and projection operations were extended to handle fuzzy conditions. Petry [30] and Yazici and George [36] overview the fuzzy database models. They survey the extensions to the relational and the object-oriented database models to develop fuzzy database models.

Some research studies have also been done for handling impreciseness in the object-oriented databases for various applications, i.e., [5], [8], [10], [11], [12], [14], [32], [35], [36]. Umano et al. [32] define a fuzzy object-oriented database model that uses fuzzy attribute values with a certainty factor and an SQL type data manipulation language. Dubois et al. [8] use 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. In this way, they formulate the certainty of membership of an object in a class. In later studies, George et al. [10], Yazici et al. [35], and Yazici and George [36] study similarity-based fuzzy

• *The authors are with the Department of Computer Engineering, Middle East Technical University, 06531 Ankara, Turkey.*
E-mail: {koyuncu, yazici}@ceng.metu.edu.tr.

Manuscript received 16 Jan. 2001; revised 26 July 2001; accepted 4 Sept. 2001. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 113479.

object-oriented database models in which they consider both fuzziness in attribute values, objects, and class hierarchy. Another object-oriented data model representing uncertainty in addition to fuzziness is proposed by Gyseghem et al. [12], in which uncertainty is handled by generalized fuzzy sets and fuzzy information by conjunctive fuzzy sets. Inoue et al. [14] define "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 are also defined by Graham [11], in which objects are extended for handling fuzziness in two ways: First, objects are allowed to contain fuzzy sets as their attribute values that 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. Furthermore, another study by Candan and Li [5] uses fuzzy logic on multimedia database for merging similarity values in queries with multiple fuzzy predicates. An alternative semantics for partial match requirement that is capable of capturing the needs of multimedia queries is discussed in that study.

Most research efforts related to integration of the object-oriented database and knowledge base technologies ([3], [22], [24], [26], [33]) assume crisp domains and ignore uncertainty in the applications. That is, attributes, objects, classes, and rules are defined with a high degree of crispness. There are only a few studies to incorporate methods for handling impreciseness in such an integrated environment. Among these, Baldwin and Martin [1] propose a system called Frill++ that combines logic programming, uncertainty, and object-oriented methods. Frill, which is a logic language with the capability of handling uncertainty, was enhanced in a way that is similar to the object model proposed by McCabe [26]. Another research effort proposed by Ndouse [28] includes a model embedding fuzzy logic into object-oriented methodology in order to deal with uncertainty and vagueness that pervade knowledge and object descriptions in the real world. In that study, a fuzzy intelligent architecture based on the uncertain object-oriented data model, which was initially introduced by Dubois et al. [8], is proposed. In [28], the classes include fuzzy if-then rules to define knowledge and the possibility theory is used for representations of vagueness and uncertainty. Finally, a recent study proposed by Lee et al. [21] introduces a fuzzy object-oriented modeling approach in which rules with linguistic terms are used to describe the relationships between attributes. Fuzzy associations are defined for establishing relationships among classes in the proposed model. A good survey of current approaches to handle imprecise information in databases and knowledge bases is given in [29]. In that study, Parsons outlines the work carried out in the database community and the work carried out in the artificial intelligence community.

In this paper, we describe an intelligent fuzzy object-oriented database environment based on the coupling of a fuzzy object-oriented database with a knowledge-based system for the next generation information systems. As an integral part of the environment, we define and extend a fuzzy object-oriented language with declarative definitions and powerful query capability. The objective is to develop a tightly coupled environment with a flexible and powerful modeling and retrieval capability to handle complex data and knowledge with uncertainty in both databases and knowledge bases. In this environment, a fuzzy object-oriented database (FOOD) model, [35], [36], is used to handle large-scale complex data, and a fuzzy knowledge-based (FKB) system with a fuzzy inference machanism is developed for handling knowledge about application domain.

Our study differs from the previous research efforts mentioned above in a number of aspects. First of all, our study introduces a unique approach for handling uncertainty and fuzziness in both databases and knowledge bases, and an architecture for coupling a database system with a knowledge-based system. The database and knowledge-based systems are coupled through a bridge mechanism that provides interoperability. Second, as the language of the architecture, a fuzzy object-oriented database language with declarative definitions and a flexible query capability is defined and described. Third, we treat imprecise data and knowledge inherent in many knowledge intensive applications by utilizing a similarity-based object-oriented database model. In addition, we model uncertainty and vagueness in data, attributes, objects, classes, and rules. Fourth, we introduce an extension to virtual class concept with fuzzy rules. Users can define virtual classes specifying a derivation rule from existing database classes. Virtual classes can inherit their attributes directly from existing classes or can derive new attributes. Moreover, a virtual class can inherit from another virtual class and its inheritance relation can be partial as well as complete. The reason to add this capability to the environment is to satisfy different schema requirements from different users and to facilitate user query formulation more efficiently. Finally, in this study, we also focus on the implementation issues of this environment. We have developed a prototype of an intelligent fuzzy object-oriented database system (IFOOD). We chose an application, an environmental information system (EIS), as the case study in this paper, since the chosen application usually involves uncertainty and requires subjective judgements to solve problems. Therefore, examples related to EIS are used throughout this paper while presenting the features of our proposed IFOOD architecture. We think that many other knowledge intensive applications involving fuzzy information and requiring imprecise deductions and/or predictions can also be modeled by using this architecture.

The paper is organized as follows: In Section 2, we provide a brief description of the IFOOD architecture. Section 3 discusses the fuzzy object-oriented data model and the query language with a special focus on uncertainty. The fuzzy knowledge base is explained in Section 4. The techniques and issues related to the coupling approach are included in Section 5. A brief description of the implementation of the prototype system of the IFOOD architecture is given in Section 6. Section 7 concludes the paper.

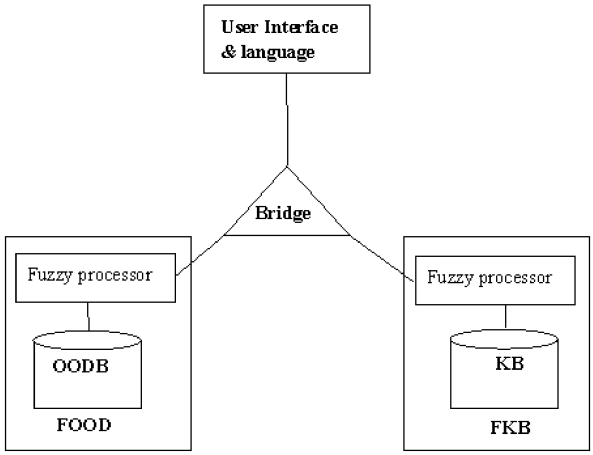

Fig. 1. The architecture of the IFOOD environment.

2 The IFOOD Architecture

The architecture of the proposed environment for intelligent fuzzy object-oriented database (IFOOD) modeling is given in Fig. 1. The architecture is based on coupling the fuzzy object-oriented database (FOOD) system [35], [36] with a fuzzy knowledge base (FKB) system and the development of a user interface and language. At the lower level of the architecture, the FOOD system serves as a database server for data management and the FKB system serves as a knowledge server for knowledge management. The communication and interaction between the database system and the knowledge base system is performed by the bridge interface. The bridge provides the desired interoperability between two systems to achieve the management of data and knowledge and allows powerful retrieval capability.

At the higher level, there is a single user interface that provides a unified environment for both data and knowledge management and allows users to have the capability of query processing independent from the physical structure of the architecture.

Fuzzy processors are used to handle uncertainty at both the object-oriented database system and the knowledge base system. At the user interface level, users are able to define objects and rules having uncertain properties and to query the system with uncertain conditions. Uncertainty issues are handled through the fuzzy processors. We have solutions to the following issues:

1. handling uncertain type definitions; fuzzy, range, and null;
2. definition of fuzzy domains, membership functions, and similarity relations;
3. calculations of membership degrees of object/class, class/superclass;
4. definition of fuzzy rules; and
5. evaluation of user queries having uncertain conditions.

The definitions of domains, similarity relations, and membership functions are stored in the object-oriented database. All the processes related to uncertainty such as object/class membership degrees or class/superclass membership degrees are done by the system automatically. Users do not have to define class-specific definitions for uncertainty except uncertain type definitions.

In the IFOOD architecture, the FKB system is a natural extension of the FOOD system. A class definition includes the definition of inheritance, attributes, methods, and rules. The FKB system processes rules while the FOOD system processes the objects, classes, etc. Since objects can have uncertain attributes and rules can have uncertain specifications, users can formulate queries with uncertain conditions. We provide required facilities in the FKB system to access to the definitions in the FOOD system. For example, if the FKB system needs the similarity of two fuzzy terms of a special domain, it gets this value via the fuzzy processors from the FOOD system. The FKB system has a fuzzy inference capability to handle fuzzy rules.

3 The IFOOD Language

In this section, we first summarize the FOOD model, initially defined in [35], [36], which is utilized in the IFOOD architecture. Then, we introduce the object-oriented database language and its use in the architecture.

3.1 The Fuzzy Object-Oriented Database (FOOD) Model

The FOOD model is similarity-based [40]. For fuzzy attributes, fuzzy domains and similarity matrices are defined. Similarity matrices are used to represent vague relationships within the fuzzy attributes. The domain, dom, is the set of values that 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, $rng \subseteq dom$. 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

$$a_i \in Attr(C) = \{a_1, a_2, \ldots, a_n\}.$$

$Attr(C)$ refers to the attributes of class C. Similar objects are grouped together to form a class. An object belongs to a class with a degree of membership. Fuzziness may occur at three different levels in the FOOD model; the attribute level, the object/class level, and the class/superclass level.

3.1.1 Attribute Level

At the attribute level, there are different types of uncertainty of the attribute values. The uncertainty type primarily considered in the FOOD model is fuzziness. Fuzzy attributes may take a set of fuzzy values, a fuzzy set, having one of the AND, OR, and XOR semantics. For example, assume that the domain of colors is:

$$dom = \{red,\ redish,\ orange,\ yellow,\ green,\ blue,\ purple,\ black,\ gray,\ white\}.$$

The following interpretations may be valid:

AND semantics: book.color = <red, blue>, i.e., "the color of the book consists of two colors, *red* and *blue*." In fuzzy sense, we may interpret these as, one of the colors is a kind of *red* and the other is more or less *blue*.

OR semantics: book.color = {redish, orange}, i.e., "the color of the book may be *redish* or *orange*." In fuzzy sense, the color of the book is vague, either *redish* or *orange* or a color similar to these.

XOR semantics: book.color = [red, blue], i.e., "the color of the book is either *red* or *blue*, but not both."

Every class has a range definition for each of the fuzzy attributes with the corresponding relevance values indicating the importance of that attribute in the definition of that class. In this way, an "approximate" description of the class can be given. An attribute of a class is allowed to take any value from a domain without considering the range values. In this model, semantics is associated with the range definitions to permit a more precise definition of a class. For instance, a class C having attributes a, b, and c from domains A, B, and C, respectively, can be defined as:

$$
\begin{aligned}
&Class\ C:\\
&rng_C(a) = \{a_1, a_3, a_6\} \quad dom_C(a) = \{a_1, a_2, a_3, \ldots, a_k\}\\
&rng_C(b) = \langle b_5, b_7\rangle \quad\ \ dom_C(b) = \{b_1, b_2, b_3, \ldots, b_m\}\\
&rng_C(c) = [c_2, c_4, c_6] \quad\ \ dom_C(c) = \{c_1, c_2, c_3, \ldots, c_n\}.
\end{aligned}
$$

Class C has "a_1, a_3, or a_6" values for the attribute a, "b_5, and b_7" requiring multivalued use of the attribute b and, finally, a value of "c_2," "c_4," or c_6" for the attribute c, only one of which is true. This is a special case of XOR; it is true only when exactly one of the entries is valid.

In the FOOD model, relevance weights are assigned to each attribute to represent the significance of the range definition of that attribute on the class definition.

3.1.2 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 ones is the boundaries of fuzzy classes being 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 one, but some objects may be related to this class with a degree between zero and one. In this case, these objects may still be considered as instances of this class with the specified degree in [0,1]. In the FOOD model, a formal range definition indicating the ideal values for a fuzzy attribute is given in the class definition. However, an attribute of an object can take any value from the related domain. So, a membership degree of an object to the class is computed by using the similarities between the attribute 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 one. Otherwise, it is either an instance with a membership degree less than one, or it is not an instance at all (when the membership degree is smaller than the threshold value) depending on the similarities between attribute values and formal range values.

To calculate the membership degree of an object to a class, we must calculate the inclusion degrees of attribute values in the range of attributes. Since the attribute values may be connected through AND, OR, or XOR semantics, the inclusion value depends on the attribute semantics.

The more similar an object's attribute values to the range definitions, the higher the class/object membership degree. But, how is this distance determined? The membership degree of an object o_j to a class C is determined by the following formula:

$$\mu_C(o_j) = \frac{\sum INC(rng_C(a_i)/o_j(a_i))^* RLV(a_i, C)}{\sum RLV(a_i, C)},$$

where $INC(rngC(a_i)/o_j(a_i))$ is the value of the inclusion taking into account the semantics of multivalued attribute values (as will be described below), $RLV(a_i, C)$ is the relevance of attribute a_i to the class C and is given in the class definition. The weighted-average is used to calculate the membership degrees of objects. All attributes, therefore, affect the membership degrees proportional to their relevance values.

3.1.3 Computation of Inclusion Values

The formulas used to calculate inclusion degrees are briefly summarized below for the AND, OR, and XOR semantics. If $o_j(a_i) = \emptyset$, then $INC = 0$ for all semantics, where o_j is an object and a_i is an attribute of the object o_j. Otherwise:

1. *AND semantics*: AND semantics requires that all of the instances exist simultaneously. 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 below:

$$
\begin{aligned}
&INC(rng(a_i)/o_j(a_i)) =\\
&\quad Min[Min[Max(\mu_S(x, y))], Min[Max(\mu_S(z, w))]],\\
&\quad \forall x \in rng(a_i), \forall y \in o_j(a_i), \forall z \in o_j(a_i), \forall_w \in rng(a_i).
\end{aligned}
$$

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

$$
\begin{aligned}
&INC(rng(a_i)/o_j(a_i)) =\\
&\quad Min[Max(\mu_S(x, z)), Threshold(o_j(a_i))],\\
&\quad \forall x \in o_j(a_i), \forall z \in rng(a_i).
\end{aligned}
$$

   The threshold value indicates the minimum level of similarity between the elements of an object attribute and it can be formulated as follows:

$$Threshold(o_j(a_i)) = Min[\mu_S(x, z)], \forall x,\ \forall z \in o_j(a_i).$$

3. *XOR semantics*: XOR semantics forces only one of the entries in the range to be true. Assuming equal probabilities for the elements of an attribute value,

the inclusion degree is computed by using the following formula:

$$INC(rng(a_i)/o_j(a_i)) = Avg[Max(\mu_S(x,y))], \quad \forall x \in o_j(a_i), \forall y \in rng(a_i).$$

The XOR semantics is much stronger than the OR semantics and forces the objects to have one of the values in the range, that is, only one of the elements in the set can be true at a time.

The next important relationship is between a class and its superclass(es), in other words, the answer of the question: "To what extent the class belongs to its superclass(es)." The FOOD model has the formulation to calculate the membership degree of a class to its superclass(es). For the details of the FOOD model including examples of the computation of inclusion values, readers should refer to the references [35] and [36].

3.2 The IFOOD Language

The IFOOD language is an object-oriented database language extended with declarative rules. The language includes basic building blocks to define and manipulate the objects. The declarative rules are used for derived classes, derived attributes, and integrity constraints. The language is enhanced with additional capability for handling uncertainty. In this section, we mainly focus on uncertainty related components of the language.

3.2.1 Uncertain Types

The IFOOD language of the architecture includes atomic types such as integer, real, and complex types such as object type and set type, which are not discussed further. There are three types for representing various uncertainties in the language. These are null type, range type, and fuzzy type.

Null type: The domain of a null type is an extension of a basic crisp domain with the three null values: "unknown," "notexist," or "nil." For example,

$$\textbf{\textit{define}}\ soilType : nullstring;$$

The *soilType* attribute can have either a string value or one of the three null values as follows:

$$soilType = "loam";\ or\ soilType = unknown;$$

Range type: Range type is used to represent incompleteness (a type of uncertainty). It is defined with two values from the same type. Values within the range must be countable and ordered. For example,

$$\textbf{\textit{define}}\ population : range\ integer;$$

The population attribute can have integer range values.

$$population = 9,500 - 10,000;$$

Fuzzy type: Fuzzy type is another type of uncertainty that is used to represent the descriptive form of uncertain information. This type has either an atomic value or a fuzzy set with various semantics (i.e., "AND," "OR," or "XOR" semantics, as mentioned before).

For example, a fuzzy type named *fuzzyTemp* is defined as follows:

$$\textbf{\textit{fuzzyOR}}\ integer\ fuzzyTemp\ \{hot, mild, normal, moderate, low, frigid\};$$

An object attribute defined with this type has either an integer value or a set of fuzzy values specified in the type definition. As an example, the *temperature* attribute can be defined and initialized as follows:

$$\textbf{\textit{define}}\ temperature : fuzzyTemp;$$
$$temperature = [hot, mild];$$

3.2.2 Term Definitions

The terms are important to understand the structures of the language. A term in the IFOOD language is defined as follows:

- A variable is either a value term or a simple object term.
- A value, v, is a value term.
- $o.a$ is a field term, where o is an object ID and a is the object attribute.
- An object identifier is a simple object term.
- $o.a(v, [thresh_{rng}(a)])$ is an object term, where o is an object ID, a is an object attribute, v is a value, and $thresh_{rng}(a)$ is the threshold level defined for the attribute a and it is optional. The threshold level is taken one (1) if it is not specified explicitly by the user.
- $c(o, [thresh_c(o)])$ is a class term, where c is the class name, o is the object ID, and $thresh_c(o)$ is the object membership level to the class c and it is optional.
- A term is a ground term if it has no variable. Fuzzy values are used in square brackets.

For example,

- $o_1.dose$ is a field term.
- o_1 is a simple object term.
- $o_1.dose([veryHigh, high]), 0.7)$ is an object term.
- $X.dose([veryHigh, high]), 0.7)$ is an object term, where X is a variable (variables start with a capital letter).
- $X.dose(Y)$ is an object term, where threshold value is omitted; that is, it is one (1).
- $pollutants(X, 0.8)$ is a class term, where X is a variable.
- $pollutants(o_1, 0.8)$ is a class term.
- $pollutants(X)$ is a class term, where threshold value is omitted.

3.2.3 Classes

In our architecture, classes are defined by using class predicates. A class definition includes class name, inheritance list, attribute descriptions, rule descriptions, and method descriptions. A class C_i is an imprecise class if at least one of its attributes is defined with an uncertain type, null, range, or fuzzy. We illustrate this with an example, rather than giving the complete syntax and semantics. Let us consider the specification of the *pollutants* class in an environment information system.

```
Class pollutants;
    define location: string;
    define dose: fdose [veryHigh, high, medium] 0.9;
    define exposureTime: ftime [veryLong, long,
           medium] 0.8;
    define contaminant: fcont [arsenic, barium,
           cadmium, mercury] 1;
    define trend: fuzzytrend [increasingSharply,
           increasing] 0.9;
    define wasteType: string;
    define structure: nullString;
    define sourceType: nullString;
    defrule X.status([dangerous],Y):- pollutants(X),
        X.dose([veryHigh, high]),
        X.exposureTime([veryLong]),
        X.contaminant([arsenic, cadmium, mercury]);
endclass;
```

The *pollutants* class has different attributes defining various properties of the objects. We extend the normal attribute definitions to handle uncertainty as follows:

```
define <attribute name> : <type>
   <attribute template> <relevance>;
```

where <*attribute name*> is an identifier that identifies an attribute of an object, and <*type*> defines a type of an attribute. <*Attribute template*> and <relevance> are two new extensions to the object-oriented database model to handle uncertainty. <*Attribute template*> is a partial set on a fuzzy domain, which indicates ideal values for a class. However, an object can take any value from the related domain. <*Relevance*> is a real value between zero (0) and one (1) that denotes the weight of an uncertain attribute in determination of the boundary of an uncertain class. <*Attribute template*> and <*relevance*> are used to calculate the membership degree of an object to its class(es) and the membership degree of a class to its superclass(es). In the given object definition, the *dose* attribute is defined as *fdose*. *fdose* is a user-defined fuzzy type and it must be defined before the class definition. The set consisting of the fuzzy terms *veryHigh*, *high*, and *medium*, is an attribute template. The relevance value of the attribute *dose* is 0.9.

New attributes may be derived by defining deductive rules in a class definition. The *status* rule, for example, which derives the actual state of the pollutants from existing attributes, is associated to the class definition. The attributes of uncertain types can be used in rule definitions. For example, *dose* is a fuzzy attribute that has fuzzy values. Y is the membership degree of the rule conclusion that is calculated from antecedent clauses. More formally, a rule is of the form $H : -L_1, L_2, \ldots, L_n$, where the head, H, is a nonnegative object term and the body $L_1, L_2, \ldots, L_n, n \geq 0$, is a sequence of class and object terms. More information about fuzzy rule definition will be given later in Section 4.

The IFOOD language provides constructs to create, update, and delete objects. The objects of the defined classes are created using the following structure:

```
object <object ID> : <class name> <field-value list>;
```

where, <field-value list> is composed of field name and value pairs. <Object-ID> is the unique identifier assigned to the object. For example, an instance of the class *pollutants* is created as follows:

```
object p1: pollutants, dose=[high],
    exposureTime=[veryLong, long],
    contaminant=[mercury],
    wasteType="factory-waste", structure=unknown,
    sourceType=unknown;
```

Instances can be deleted or updated using *delete* and *update* commands, respectively, with the following syntax:

```
delete<object ID>;
update <object ID>, <field-value list>;
```

where, <field-value list> is the same structure used in the object creation.

3.2.4 Virtual (Derived) Classes

The object-oriented database model organizes the objects into a class hierarchy. Objects are stored into the database according to the defined class hierarchy. However, database serves many users that have different demands that may result in a complex class hierarchy, which can be hard to handle and comprehend. Defining virtual (derived) classes on the existing ones can satisfy some of the requirements efficiently. The ability to define virtual classes increases expressiveness for extending the modeling environment. Users can define different user views of the database without explicitly extending the database class schema.

A virtual class may be a specialization of a database class with the same attributes and may contain a subset of the objects of the database class. The objects may be selected by the constraint defined in a virtual class definition. For example, a virtual class that includes the *gaseous_pollutants* is defined as follows:

```
(v-1) vclass gaseous_pollutants;
          inherits pollutants;
          condition gaseous_pollutants(X):- pollutants(X),
              X.structure(gaseous);
      endclass;
```

In this example, derivation of a new class is done by a logical rule. The header of the rule is a class term specifying a new virtual class. Only the objects being in the gaseous structure are derived as instances of the new class.

The relationship between a virtual class and its superclass(es) may be fuzzy as it is in the database schema and the objects of this virtual class may be instances of this class with a partial membership. For such cases, the uncertain attributes can be redefined with their template and relevance values to determine class/superclass and object/class membership degrees as shown below:

```
(v-2) vclass dangerous_pollutants;
          inherits pollutants;
          condition dangerous_pollutants(X):-
              pollutants(X,0.6);
          define dose: fdose [veryHigh] 1;
          define exposureTime: ftime [veryLong] 0.8;
          define contaminant: fcont [arsenic, barium,
              cadmium, mercury] 0.9;
      endclass;
```

The *dangerous_pollutants* class is a subclass of the *pollutants* class. Its uncertain attributes are redefined in the attributes section with new attribute template and relevance values. The objects of the *pollutants* class, which have a membership degree equal to or greater than 0.6, are derived as instances of the *dangerous_pollutants* class. The threshold level for an object may be given by the user to determine the level of the object selection as it is in the given example above. If it is not specified, it is taken as one by default. If fuzzy attribute template and relevance values are not specified, the superclass definitions are inherited. Class/superclass and object/class membership degrees are computed as explained in [35], [36].

Furthermore, a virtual class may inherit attributes from different classes to form different views of the objects (multiple inheritance). Derived attributes can be defined in virtual classes. In this way, it is possible to create new views by deducing or calculating new attributes from existing ones, by renaming existing attributes of classes. A new class, named *pollution*, is defined in the following example. *Pollution* is formed when pollutants are dangerous for sites such as city, water source, etc. The *pollution* class is defined to determine whether the pollutants form pollution for the sites.

(v-3) ***vclass*** *pollution*;
inherits *pollutants, sites*;
condition *pollution(Z):- pollutants(X),*
X.dose([veryHigh, high]),
X.exposureTime([veryLong]),
X.contaminant([arsenic, cadmium, mercury]),
X.location(A), sites(Y), Y.location(A);
endclass;

In this example, the pollution class inherits from two different database classes, *pollutants* and *sites*. The rule derives new objects from two classes. The IFOOD language includes constructs to manipulate virtual classes and also supports the virtual class queries, which will be explained in Section 5.

3.2.5 Additional Definitions for Handling Uncertainty

Fuzzy and crisp types are handled together by utilizing the membership functions and the similarity relations. It is also possible to query the database both with fuzzy and crisp conditions. In order to handle both crisp and fuzzy values uniformly, the membership functions are used to determine the fuzzy set to which the crisp values belong. We associate a membership function to each fuzzy term defined in the system as follows:

membershipof *<ftype> <fterm>*
<function declaration>;

A number of predefined membership functions are available in the environment and the user can add new membership functions if needed. For example, the membership function definition of the fuzzy term *hot* that belongs to the *fuzzyTemp* type is given below:

membershipof *fuzzyTemp hot triangle 25,40,55;*

Another definition related to uncertainty is the similarity relation. Informally, similarity relation S is a fuzzy relation that defines similarities between every pair of the elements in a fuzzy domain D [40]. More formally, a similarity relation $\mu_S(x, y)$, for a given domain, D, is a mapping of every pair of elements in the domain onto interval [0,1]. A similarity matrix is defined in the system as follows:

simmat *<ftype> <matrix entries>;*

where, *<matrix entries>* is a real list whose elements are in between zero and one. For example, the fuzzy domain for *temperature* is *{hot, mild, normal, moderate, low, frigid}* and the fuzzy type defined for *temperature* is *fuzzyTemp*. The similarity relation for *fuzzyTemp* can be defined as follows:

Simmat *fuzzyTemp [1.0, 0.7, 0.2, 0.2, 0.2, 0.2,*
0.7 1.0 0.2 0.2 0.2 0.2,
0.2 0.2 1.0 0.7 0.7 0.7,
0.2 0.2 0.7 1.0 0.7 0.7,
0.2 0.2 0.7 0.7 1.0 0.7,
0.2, 0.2, 0.7, 0.7, 0.7, 1.0];

The similarity relation and the membership functions of a domain are related to each other and they are defined in a consistent manner to eliminate the potential for inconsistency in the system. The inconsistency may cause unexpected conclusions later in the queries. Therefore, a consistency check should be done at the definition phase of the similarity relations and the membership functions.

3.2.6 Object Queries

We use an SQL-like query syntax in which declarative formulas are used, i.e., we adopt a ***select ...from ...where*** construct for extracting information from database. A complete IFOOD query can be formed by using the following syntax:

select *field terms* ***from*** *class terms* ***where*** *object terms;*

Rather than giving the complete specification of the query language, we only discuss the uncertain querying along with examples. Users are able to form uncertain queries as follows:

select *X.location* ***from*** *pollutants(X, 0.7)*
where *X.dose([veryHigh],0.6),*
X.exposureTime([veryLong],0.8);

where $pollutants(X, 0.7)$ gets the pollutant objects which have a membership degree equal to or greater than 0.7 (a threshold value specified in the query), and $X.dose([veryHigh], 0.6)$ selects the objects that have a very high *dose* with a similarity or membership degree equal to or greater than 0.6 (it is also a threshold value specified in the query). $X.exposureTime([veryLong], 0.8)$ is another fuzzy condition to be evaluated like the *dose* attribute. Notice that the statements in ***from*** and ***where*** parts are expressed in a declarative way. The defined declarative rules (derived attributes) can be used in the queries. For example, consider the following *status* rule:

X.status ([dangerous],Y):-pollutants(X),
X.dose([veryHigh,high]),

X.exposureTime([veryLong]),
X.contaminant([arsenic, cadmium, mercury]);

where Y is for the membership degree of the rule. This rule can be used in a query as follows:

select *X.location* ***from*** *pollutants(X, 0.7)*
where *X.status([dangerous],0.8);*

In this query, the pollutants having an object membership degree greater than or equal to 0.7 are selected and then the *status* rule is fired and the objects whose *status* are dangerous with a membership degree 0.8 or above are selected. Notice that the *status* rule was defined as a derived attribute in the class definition. The fuzzy rules will be explained in Section 4 and more examples about query processing will be given in Section 5.

4 Fuzzy Knowledge Bases (FKB)

In this section, we describe the FKB model utilized in the IFOOD architecture. The KB system used in the IFOOD architecture includes intelligent objects having fuzzy attributes and rules. In addition, a fuzzy inference method used for deduction of fuzzy conclusions is introduced.

4.1 Structure of Fuzzy Rules

In the FKB, knowledge is represented by *IF-THEN* rules in which the antecedent and the consequent involve linguistic variables. For example,

IF x is A THEN y is B

where x and y are linguistic variables and A and B are fuzzy sets. The antecedent of a rule may be a composite of more than one clause connected by the fuzzy logical operators AND and OR. For example,

IF dose is high OR dose is veryHigh
AND exposureTime is long
THEN pollution is veryDangerous.

Such fuzzy rules can be defined to derive new attributes or to specify some constraints using not only crisp but also fuzzy attributes of different objects. The variables of rules represent attributes of objects or objects themselves. For example, consider the domain of the following fuzzy attributes of the *pollutants* class.

domain(dose)={veryHigh, high, medium, low}.
domain(exposureTime)={veryLong, long, medium, short}.
domain(status)={extremelyDangerous, veryDangerous, dangerous, lessDangerous}.

The fuzzy rules can be defined using these linguistic values as follows:

IF pollutants.dose is veryHigh
AND pollutants.exposureTime is veryLong
THEN pollutants.status is veryDangerous.

The rule given below exemplifies the exact syntax of the fuzzy *IF-THEN* rules utilized in the IFOOD language.

(R-1) defrule *X.status([veryDangerous],Y):- pollutants(X), X.dose([veryHigh]), X.exposureTime([veryLong]);*

where $X.status([veryDangerous], Y)$ is the consequent of the rule (the THEN part or the left-hand side (LHS) of the rule) and the right-hand side (RHS) of the ":-" sign is the antecedent of the rule (the IF part). The IFOOD rules are constructed utilizing the class and object terms of the IFOOD language, which are described in Section 3.2. In the consequent of the rule, Y is the membership degree of the rule conclusion, which is computed by using the matching degrees of the rule antecedent conditions. The method for calculation of Y will be described in the coming section.

4.2 Similarity Matching

In our architecture, we assume that there may not be a strict border among fuzzy terms. We use similarity relations to define similarities between pairs of elements in the fuzzy domain. Even if there is no exact matching, similar rule(s) may still be activated in fuzzy systems. A rule's antecedent with a nonzero membership value may cause the rule to be activated. In order to eliminate undesired effects and increase the efficiency of data retrieval, a threshold value is employed. The membership grade of the consequent of a rule is obtained from the matching degrees of the antecedent conditions of the rule. Similarity matching values between two fuzzy terms are determined by using the predefined similarity relations of the fuzzy terms. How we define the similarity relations is described in Section 3.2. The values of the similarity relations can be defined either by domain experts or be computed using different methods [34], [39]. Furthermore, applications can start with the initial definitions of similarity relations and these definitions can be improved considering the experience acquired from the operation of the applications.

For example, consider the rule (R-1) given above. If there is any *pollutant*'s object having values, which exactly match with the rule antecedent (i.e., *dose* is *veryHigh* and *exposureTime* is *veryLong*), this rule will succeed. However, if the values of an object do not exactly match with any rule's antecedent, there still may be a possibility for similarity matching. Here, by similarity matching, we mean that not only the rule with exact matching is activated but also any other rule may be activated when the corresponding fuzzy values are similar.

For example, assume that the following values are given:

$$\mu_S(high, veryHigh) = 0.8;\ Threshold = 0.7;$$
$$Object.dose = [high].$$

Even though the value of *dose* is *high*, instead of *veryHigh*, that condition of this rule (R-1) is satisfied with a matching degree of 0.8 since the similarity of *high* to *veryHigh (i.e., 0.8)* is greater than the priori given threshold value (i.e., 0.7).

Since the IFOOD language permits us to represent both fuzzy and crisp values, some objects may have fuzzy values and some may have crisp values for the same attribute. If the value of object attribute is fuzzy, the rules are activated using similarity matching as explained above. If the value of object attribute is crisp, then the membership degree of a crisp value to the fuzzy set in the rule is determined by using a priori defined fuzzy membership function. If that membership value, which is in [0,1], is greater than or equal

to the specified threshold value, then the rule condition is satisfied.

A rule's antecedent may be composed of more than one condition. Each predicate in a rule antecedent may have its own matching degree with a corresponding object value. Therefore, we need to calculate an overall matching degree of the rule antecedent. Here, for simplicity, we use the min operator for combining the degree of matching of conjunction (AND) conditions, and the max operator for combining the degree of matching of disjunction (OR) conditions, as shown below:

For AND operator:

$$\mu_{antecedent} = Min(\mu_1, \mu_2, \ldots, \mu_n).$$

For OR operator:

$$\mu_{antecedent} = Max(\mu_1, \mu_2, \ldots, \mu_n).$$

Considering the rule (R-1) given above, each object term is matched with a matching degree (i.e., $X.dose([veryHigh])$ and $X.exposureTime([veryLong])$). In addition to object terms, the rule also includes a class term (i.e., *pollutants(X)*). The matching degree of this term is the object membership degree, which shows the membership of the object to the related class. It is calculated as described in Section 3.1 during the object creation and stored with the object in the database. This matching degree is directly obtained from the database when needed.

Another issue related to the rule definitions is the usage of fuzzy and crisp attributes together in the rule definitions. If an antecedent predicate is defined using a crisp attribute, then the traditional pattern matching is applied and the matching degree of this predicate is one (1) in case of successful matching, otherwise the rule fails.

4.3 The Fuzzy Inference Method

Fuzzy implication rules are generalizations of implications in two-valued logic. The inference of fuzzy implications generalizes two kinds of logic inference using the implications in classical logic: *modus ponens* and *modus tollens* [39]. However, we use the similarity matching *modus ponens* for fuzzy implications in the knowledge base. Fuzzy inference machanism produces a conclusion that is qualified and quantified. The conclusion is qualified using *modus ponens (MP)* inferencing method as follows:

Exact matching MP :

Rule : x is $A \rightarrow y$ is B
Fact : x is A
———————————
Infer : y is B

Similarity matching MP:

Rule : x is $A \rightarrow y$ is B
Fact : x is A'
———————————
Infer : y is B'

where $\mu_S(A', A) > 0$. Similarity matching is generalized form of exact matching.

Fuzzy inference machanism quantifies the conclusion with a membership degree using an implication function. There are different implication functions proposed in literature [39]. We utilize the Goguen's fuzzy implication function in the FKB system.

$$t(x_i \; is \; A \rightarrow y_i \; is \; B) = \begin{cases} 1 & \mu_A(x_i) \leq \mu_B(y_i) \\ \frac{\mu_B(y_i)}{\mu_A(x_i)} & \mu_A(x_i) > \mu_B(y_i) \end{cases} \quad (1)$$

For example, assume that the similarity of *high* to *veryHigh* is given as 0.8 and similarity of *dangerous* to *veryDangerous* is given as 0.7.

Rule : IF pollutant.dose is *veryHigh* THEN
pollutant.status is *veryDangerous*.
Fact : pollutant_1.dose is *high*.
———————————
Infer : pollutant_1.status is *dangerous*.

The matching degree of antecedent is:

$$\mu_{dose}(high, veryHigh) = 0.8.$$

The matching degree of consequent is:

$$\mu_{status}(dangerous, veryDangerous) = 0.7.$$

Since the matching degree of antecedent (i.e., 0.8) is greater than the matching degree of consequent (i.e., 0.7), the rule conclusion is calculated as follows:

$$\mu_{conclusion} = 0.7/0.8 = 0.87.$$

Therefore, the status of the $pollutant_1$ object is *dangerous* with a membership degree 0.87.

In the given example, the rule's antecedent includes only one predicate. If a rule antecedent consists of more than one predicate, then the overall matching degree of the rule's antecedent is calculated as explained in the previous section.

Since all the similar rules will be activated during fuzzy inferencing, we combine the conclusions of the activated rules to produce the final output of the inference. We combine the conclusion by using the maximum operator [39] as follows:

$$\mu_{FINAL} = Max(\mu_{R1}, \mu_{R2}, \ldots, \mu_{RN}).$$

4.4 The Fuzzy Inference Engine Model

The inference engine is the inference and control mechanism of a knowledge-based system. It gets the rules and facts/objects as input, tries to satisfy rules comparing with facts, and produces a conclusion from the satisfied rules. This process of inference engine continues as a cycle.

In a crisp knowledged-base system, typically, the inference engine has a cycle consisting of three phases: matching, selection, and firing/execution. At the matching phase, satisfaction of the antecedent of the rules is tested by examining the working memory (facts/objects). At the selection phase, the activated rule with the highest priority according to the implemented conflict resolution strategy is selected for execution. At the firing phase, the consequent of the selected rule is processed.

For a fuzzy knowledge-based system, the inference engine needs to be modified. The fuzzy inference engine model of the IFOOD system is shown in Fig. 2. The

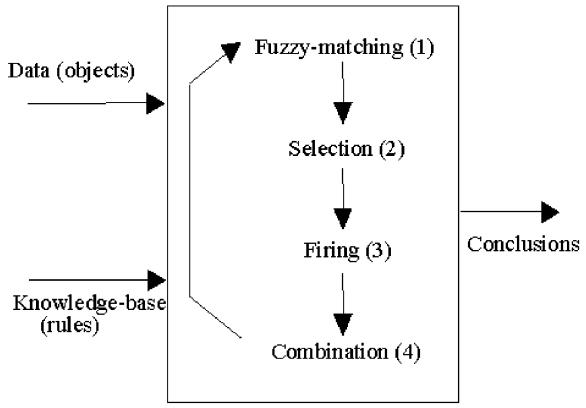

Fig. 2. Inference engine model of the IFOOD system.

inference has an inferencing cycle consisting of the fuzzy-matching phase, the selection phase, the firing phase, and the combination phase.

The phases 1, 2, and 3 basically perform the actions done by a crisp inference engine. However, one new phase is added. In addition, if the inference engine should produce crisp outputs from the fuzzy rules, then a defuzzification should also be added as the fifth phase. In this study, we combine forward and backward reasoning. The matching phase works as forward chaining to determine applicable rules. The execution of the rules is done according to the backward chaining method by selecting related rules to prove user query (goal), which includes the consequence of the rule. However, execution does not stop when the goal is satisfied, but continues until all possible solutions are found. Then, all possible solutions are combined to produce a single conclusion. Selection, execution, and combination phases are integrated to each other. The task of each phase in a fuzzy inference engine can be summarized as follows:

- *Fuzzy-matching phase*. In this phase, the inference engine tries to satisfy the rule's antecedent conditions according to fuzzy matching algorithm. Therefore, this phase performs the similarity matching as explained above. If the matching is successful, then a matching degree is computed. When all the conditions of the rule's antecedent are satisfied, the matching degree of the rule antecedent is calculated by the method as described in Section 4.2.
- *Selection phase*. The rules whose antecedents match are passed to the second phase and they are the candidates for execution. There may be more than one rule in the conflict set as the activated rules. In case of fuzzy inferencing, the number of the activated rules may increase since not only the exact matching rules will be selected but also partially matching rules (similar rules) may be activated. The task of this phase is to determine the rules that may likely satisfy the user query and to select those rules for execution in the appropriate order.
- *Firing phase*. In this phase, the membership degree of the rule conclusion is calculated according to the employed implication function. This phase performs the process explained in the previous section (Section 4.3). Then, the consequent of the rule is processed to produce a conclusion and the obtained conclusion is put in the conclusion list.
- *Combination phase*. The inference engine combines the conclusions inferred by all similar rules into a final conclusion. Since a particular input to the system often triggers multiple fuzzy rules, a combination phase is needed. The inference engine gets the most significant conclusion applying the *max* fuzzy disjunction operator to the conclusion list acquired from the previous phase. This conclusion will be the output of the fuzzy inference engine of the FKB system.

5 Coupling FOOD with FKB

5.1 Bridge Functionality

At the lower level of the architecture, we have the FOOD system for data management and the FKB system for knowledge management. The communication between the database and the knowledge base components is through an interface that is called the bridge. The bridge provides the desired interoperability for accommodating data and knowledge management together.

We provide an integrated language to manage data and knowledge at the user-interface level. There are two different, but coupled systems, at the architecture. A class definition may include attributes, inheritance list, method definitions, and also rule definitions. We can summarize the tasks of the bridge mechanism as follows:

- Fuzzy data related to fuzzy types, membership functions, and similarities are stored in the FOOD system. The bridge gets data from user interface and passes to the FOOD system in an appropriate format. On the other hand, the FKB system has fuzzy inference mechanism. However, the data related to fuzziness are stored in the FOOD system. The FKB needs fuzzy data stored in the FOOD system for fuzzy inferencing. Therefore, the bridge provides interoperability for the exchange of crisp and uncertain data/information between the FOOD and the FKB systems.
- In the IFOOD architecture, the FKB system handles the rules and the rest of the class and objects are handled by the FOOD system. The bridge gets class definitions from user interface/language and sends them to the FKB and/or to the FOOD system in the appropriate formats. It is not possible to connect rule definitions directly to a class definition in the existing systems since they are two separate systems. We use a catalog to handle this situation. The bridge maintains this catalog to handle extra properties that are not possible to handle directly in the underlying database or the knowledge base systems.
- The FKB system handles virtual (derived) classes to satisfy different query requirements. Virtual classes are connected to the base class schema, which is maintained by the FOOD system, through one of the specialization, generalization, aggregation, or association relationships. Therefore, the bridge that

TABLE 1
Objects in the Fuzzy Object-Oriented Database

objectID	class	dose	exposureTime	location	structure	trend	μ
p1	Pollutants	veryHigh	veryLong	Ankara	solid	stable	1
p2	Pollutants	low	short	Istanbul	solid	decreasing	0.4
p3	Pollutants	high	medium,long	Ankara	gaseous	increasing	0.8
p4	Pollutants	veryLow	veryShort	Bursa	gaseous	stable	0.2
p5	Pollutants	medium, high	short	Izmir	liquid	increasing	0.7

TABLE 2
Similarity Relation of Dose

Fuzzy OR integer fuzzy Dose {veryHigh, high, medium, low, veryLow}

dose	veryHigh	high	medium	low	veryLow
veryHigh	1.0	0.9	0.7	0.2	0.2
high	0.9	1.0	0.7	0.2	0.2
medium	0.7	0.7	1.0	0.2	0.2
low	0.2	0.2	0.2	1.0	0.5
veryLow	0.2	0.2	0.2	0.5	1.0

provides interoperability between two systems also keeps the relationship between a virtual class and the base class schema.

- Another important task of the bridge mechanism is to coordinate processing of the user queries. Users can formulate queries including fuzziness and rule firings without considering any underlying detail of the system, neither the FOOD system nor the FKB system. The bridge has a coordinator role in query evaluation. This issue will be explained in more detail with examples in the coming sections.

Bridge consists of several functions to perform these tasks. For example, the class creation function can be defined by the following pseudocode:

```
BEGIN CLASS-CREATION
   Get (class-definition);
   Parse (class-definition);
   Insert-to-OODB (class-definition);
   Insert-to-KB(class-definition);
   IF class-definition includes rule-definition(s) THEN
      Insert-to-KB(rule(s));
   Save-to-catalog(class-definition);
   Commit-changes();
END CLASS-CREATION
```

5.2 Query Evaluation

The bridge gets user queries, analyzes them, sends requests to the FOOD system and/or to the FKB system, retrieves the results, and sends them up to the user interface. Therefore, the bridge has an important role in the evaluation of queries. Sometimes, a query may be very simple like getting a crisp object attribute from database. Sometimes, it may be very complex requiring fuzzy evaluation and deduction. The bridge reforms user queries in an appropriate format for the underlying systems. For rule firings in the FKB system, the bridge transfers the objects from the FOOD system to the FKB working memory. Objects are stored in the FOOD system and only the required parts are transferred from the FOOD system to the FKB working memory for evaluation. The process of the query evaluation done by the bridge mechanism is given below as a pseudocode:

```
BEGIN QUERY-EVALUATION
   Get(query);
   Parse(query);
   IF query (from statement) includes virtual class
      THEN
         Evaluate(virtual-class-condition);
         Get(satisfied-objects);
   ELSE
      Construct(crisp-query);
      Send-to-OODB(crisp-query);
      Get(satisfied-objects);
   IF query includes fuzzy predicate(s) THEN
      Start-fuzzy-evaluation(satisfied-objects);
      Get(satisfied-objects);
   IF query includes knowledge-base predicate(s)
      (rule) THEN
      Transfer-to-knowledge-base(satisfied-objects);
      Start-inference-engine-evaluation();
      Get(satisfied-objects);
   Submit-to-user(satisfied-objects);
END QUERY-EVALUATION
```

We will consider four different cases to further clarify the functionality of the bridge mechanism. The example objects used in these cases are included in Table 1. The fuzzy type definitions and similarity relations are given in Table 2 and Table 3, respectively, for the attributes *dose* and *exposureTime*:

Case 1. The following query requests the object locations having a membership degree greater than or equal to 0.7 and having *veryHigh dose* and *veryLong exposure time* with the specified levels.

TABLE 3
Similarity Relation of ExposureTime

Fuzzy OR integer fuzzy Exptime {veryLong, long, medium, short, veryShort}

exposure-time	veryLong	long	medium	short	veryShort
veryLong	1.0	0.9	0.8	0.6	0
long	0.9	1.0	0.8	0.6	0
medium	0.8	0.8	1.0	0.6	0
short	0.6	0.6	0.6	1.0	0
veryShort	0	0	0	0	1.0

select *X.location* ***from*** *pollutants(X, 0.7)*
where *X.dose([veryHigh],0.6),*
X.exposureTime([veryLong],0.8);

In this query, *dose* and *exposureTime* are attributes of the *pollutants* class. This is a query directly related to the FOOD system. In this case, the bridge gets this query and transmits it to the FOOD system in the appropriate format. Then, the query is evaluated as follows:

- The first predicate to evaluate in this query is $pollutants(X, 0.7)$. This predicate selects the *pollutants* objects having a membership degree greater than or equal to 0.7. The object membership degrees computed and the results are included at the last column in Table 1. Interested readers can refer to [35] and [36] for object membership computations. The objects $p1$, $p3$, and $p5$ satisfy this condition, since their membership values are greater than or equal to the threshold value, 0.7, which is specified in the query.
- Then, the predicate $X.dose([veryHigh], 0.6)$ should be evaluated.
  - p1.dose is veryHigh.
    $$\mu_S(veryHigh, veryHigh) = 1;$$
    therefore, p1 satisfies.
  - p3.dose is high. $\mu_S(veryHigh, high) = 0.9$; therefore, p3 satisfies.
  - p5.dose is medium OR high.
    $$Max(\mu_S(veryHigh, high), \mu_S(veryHigh, medium)) = Max(0.9, 0.7) = 0.9.$$
    Therefore, p5 satisfies.
- Finally, the predicate
  $$X.exposureTime([veryLong], 0.8)$$
  should be evaluated.
  - p1.exposureTime is veryLong.
    $$\mu_S(veryLong, veryLong) = 1;$$
    therefore, p1 satisfies.
  - p3.exposureTime is medium OR long.
    $$Max(\mu_S(veryLong, long), \mu_S(veryLong, medium)) = Max(0.9, 0.8) = 0.9.$$
    Therefore, p3 satisfies.
  - p5.exposureTime is short. $\mu_S(veryLong, short) = 0.6$; therefore, p3 does not satisfy.
- At the end of the fuzzy query evaluation, the objects $p1$ and $p3$ satisfy the user query conditions. Then, these objects are retrieved and submitted to the user interface as the answer to the query by the bridge mechanism.

Case 2. In this case, *status* is a rule defined in the class definition as seen below. We call that as a derived attribute since it is defined to derive status information from the existing attributes of the objects. This rule is handled by the knowledge base system.

defrule *X.status([veryDangerous],Y):- pollutants(X),*
X.dose([veryHigh,high],0.7),
X.exposureTime([veryLong],0.7);

The following query requests the pollutants that have a very dangerous status. This query causes to fire the *status* rule in the knowledge base.

select *X.location* ***from*** *pollutants(X, 0.7)*
where *X.status([veryDangerous],0.8);*

In this case, the query needs a rule firing since *status* is defined on the knowledge base. The bridge follows the following procedure to retrieve the answer for this query:

- It checks its catalog to understand what *status* is. *Status* is a rule defined in the knowledge base.
- It starts with transferring the required objects from the FOOD system to FKB memory. Transfer is done according to the *from* statement since there is no other statement related to the FOOD system. The predicate $pollutants(X, 0.7)$ specifies that the objects having a membership degree greater than or equal to 0.7 should be evaluated. For our example, the objects $p1$, $p3$, and $p5$ have membership degrees greater than or equal to 0.7. Therefore, these three objects are transferred from the FOOD system to the FKB working memory for further rule evaluation.
- It starts evaluation of the inference mechanism. With the first predicate $pollutants(X)$, all the *pollutants* object, which are in the FKB memory, are evaluated. Then, the second predicate,
  $$X.dose([veryHigh, high], 0.7),$$
  is evaluated as follows:

- For p1:

$$Max(\mu_S(veryHigh, veryHigh), \mu_S(veryHigh, high)) = Max(1, 0.9) = 1.$$

- For p3:

$$Max(\mu_S(high, veryHigh), \mu_S(high, high)) = Max(0.9, 1) = 1.$$

- For p5:

$$Max(\mu_S(medium, veryHigh), \mu_S(medium, high), \mu_S(high, veryHigh), \mu_S(high, high)) = Max(0, 7, 0.7, 0.9, 1) = 1.$$

These three objects satisfy this condition of the rule antecedent.

- Then, the predicate $X.exposureTime([veryLong], 0.7)$ is evaluated:
  - For p1 : $\mu_S(veryLong, veryLong) = 1$.
  - For p3: $Max(\mu_S(medium, veryLong)$,
    $\mu_S(long, veryLong)) = Max(0.8, 0.9) = 0.9$.
  - For p5: $\mu_S(short, veryLong) = 0.6$.

  The object p5 does not satisfy this condition of the rule since matching degree is less than the specified threshold (i.e., $0.6 < 0.7$).
- The overall matching degree of the rule antecedent is calculated as follows:
  - For p1: $\mu_{\text{antecedent}} = Min(1, 1, 1) = 1$.
  - For p3: $\mu_{\text{antecedent}} = Min(0.8, 1, 0.9) = 0.8$.

  Notice that the first number in the *min* statement is the object membership degree coming from the $pollutants(X)$ predicate.
- Finally, the membership degree of the rule conclusion is calculated as explained in Section 4.3. Since the matching degree of the rule consequent is greater than the matching degree of the antecedent, the rule conclusion is one for the objects p1 and p3 according to the implication formula (1) given in Section 4.3.
- The bridge gets the result from the knowledge base. The objects $p1$ and $p3$ are returned in the answer set.
- Finally, the bridge submits the result obtained from the FKB to the user interface.

Case 3. We can consider a complex situation as the third case.

select *X.location* ***from*** *pollutants(X, 0.7)*
where *X.structure(solid),X.status([dangerous],0.7);*

In this query, *structure* is a normal attribute and *status* is a derived attribute. The first predicate, *X. structure(solid)*, should be evaluated on the FOOD system and the second predicate, $X.status([dangerous], 0.7)$, should be evaluated on the FKB system. The task of the bridge becomes more complex in this case. It follows the following procedure to process this query:

- It checks its catalog to understand what *structure* and *status* are. *Structure* is an attribute that should be evaluated by the FOOD system and *status* is a rule that should be evaluated by the FKB system.
- It forms a new query including the predicates related to the FOOD system and sends that to the FOOD system. First, the predicate $pollutants(X, 0.7)$ is evaluated. The objects having an object membership degree greater than or equal to the threshold value, 0.7, are selected. For our example, the objects $p1$, $p3$, and $p5$ satisfy this condition.
- Then, the second predicate $X.structure(solid)$ is evaluated. Only the first object satisfies this condition.
- It transfers the selected objects as the result of the query evaluation of the FOOD system, to the knowledge base. For the given example, the object $p1$ is transferred from the FOOD system to the FKB memory at the end of the evaluation.
- It starts evaluation of the inference engine. The first clause $pollutants(X)$ causes all the *pollutants* object to be evaluated. Then, the second clause,

$$X.dose([veryHigh, high], 0.7),$$

is evaluated as follows:

- For p1: $Max(\mu_S(veryHigh, veryHigh)$,
  $\mu_S(veryHigh, High)) = Max(1, 0.9) = 1$.

- Then, the predicate $X.exposureTime([veryLong], 0.7)$ is evaluated:
  - For p1: $\mu_S(veryLong, veryLong) = 1$.
- Then, the overall matching degree of the rule antecedent is calculated.
  - For p1: $\mu_{\text{antecedent}} = Min(1, 1, 1) = 1$.
- Notice that, in this query, the rule consequent does not match exactly with the condition specified in the query. The rule head in the FKB is $X.status([veryDangerous], Y)$, while the query specification is $X.status([dangerous], 0.7)$. Assume $\mu_S(dangerous, veryDangerous)$ is 0.8 in the system catalog. This rule is fired for this query since the antecedent of the rule matches with the object. According to the implication formula (1) given in Section 4.3, the conclusion degree is calculated as follows:
  - For $p1$: $\mu_{\text{conclusion}} = 0.8/1 = 0.8$. Therefore, $p1$ satisfies the condition of the user query since the threshold level for the rule is specified as 0.7 in the user query (i.e., $0.8 > 0.7$).
- The result is retrieved from the knowledge base. The object $p1$ is the answer for the query.
- The bridge submits the final result of the query to the user interface.

Case 4. In this case (involves in a more complex situation), we define a new derived attribute, *action_requirement*, in which another derived attribute is used. *Status* is also a derived attribute as explained before.

defrule *X.action_requirement([urgent],Y):-*
pollutants(X), X.status([veryDangerous],0.7),
X.trend([increasingSharply],0.8);

Assume that the following query is formulated by user:

select *X.location* ***from*** *pollutants(X, 0.7)*
where *X.action_requirement([urgent],0.7);*

- For this query, first evaluation is done on the FOOD system according to the *from* statement as in the previous examples. The objects $p1$, $p3$, and $p5$ satisfy the *from* statement and they are transferred to the FKB memory for the derived attribute (*action_requirement*) evaluation.

  The rule *action_requirement* is fired on the knowledge base, which causes the firing of the *status* rule. The evaluation of the *status* rule is already given in "case-2" for these objects. According to these results, $p1$ and $p3$ satisfy the *status* condition.

  The threshold level of the *status* rule is specified as 0.7 in the *action_requirement* rule. Therefore, the objects $p1$ and $p3$ satisfy the

  $$X.status([veryDangerous], 0.7)$$

  predicate of the *action_requirement* rule.
- Evaluation of the *action_requirement* rule continues with the next predicate,

  $$X.trend([increasingSharply], 0.8).$$

  Assume that the following similarities are given:

  - For p1: $\mu_S(increasingSharply, stable) = 0.5$.
  - For p3:

  $$\mu_S(increasingSharply, increasing)) = 0.8.$$

  Since the threshold level is specified as 0.8 for this predicate, only the object $p3$ satisfies the condition. The matching degree of the rule antecedent is:

  $$\mu_{\text{antecedent}} = Min(0.8, 1, 0.8) = 0.8$$

  for $p3$ and the rule conclusion membership degree is 1 since the rule consequent exactly matches. In the min statement, 0.8 is the object membership degree inserted by the $pollutants(X)$ predicate, 1 is the membership degree of the status rule inserted by $X.status([veryDangerous], 0.7)$, and 0.8 is the matching degree of the *trend* attribute, for example, $(X.trend([increasingSharply], 0.8))$.
- The object $p3$ is submitted to the user interface as the answer at the end of the query evaluation.

5.3 Virtual Class Processing

Virtual classes, as explained in Section 3, are handled by the FKB system. The bridge keeps information related to virtual classes in the catalog. When there is a query related to virtual classes, the bridge checks the catalog and activates the required processes to produce an answer to the query. From the user point of view, there is no difference between the usage of the normal database classes and the usage of virtual classes specified in queries. Users can formulate queries including virtual classes without specifying any extra information related to virtual classes. The bridge handles coordination of query processing. Some examples may clarify further virtual class query processing. Consider the *gaseous_pollutants* virtual class, (v-1), given in Section 3.2.4 and assume the following query is formulated:

select *X* ***from*** *gaseous_pollutants(X)*
where *X.location(ankara);*

The bridge processes this query as follows:

- The bridge determines the *gaseous_pollutants* class from the catalog as a virtual class.
- It forms a database query from the predicates $pollutants(X)$ and $X.structure(gaseous)$ which specify the objects of the *pollutants* class being in gaseous structure should be selected. The objects satisfying this condition are transferred from the FOOD system to the FKB memory.
- From the database objects given in Table 1, the objects $p3$ and $p4$ are transferred from the FOOD system to the FKB memory since they are in gaseous structure. The rest of the query is processed in the FKB system. The predicate $X.location(ankara)$ is evaluated by the FKB system and the object $p3$ satisfies the overall query in this example.

Let us consider the virtual class (v-2) given in Section 3.2.4 as the second example. The virtual class is also fuzzy and it is connected to its superclass *pollutants* with a membership degree. Assume the following query, selecting the *dangerous_pollutants* class objects having very high dose:

select *X* ***from*** *dangerous_pollutants(X)*
where *X.dose([veryHigh],1).*

- At the first step, the bridge checks its catalog and determines *dangerous_pollutants* as a virtual class.
- The objects of the *pollutants* class having object membership degree equal to or greater than 0.6 are transferred from the FOOD system to the FKB system. However, the object membership degrees are recalculated according to the template and relevance values specified in the virtual class definition. Since the threshold is defined as 0.6 in the virtual class definition, the objects $p1$, $p3$, and $p5$ are transferred to the FKB in this example.
- The predicate $X.dose([veryHigh], 0.9)$ is evaluated in the FKB and the satisfying objects are submitted to the user interface at the end of the query evaluation. Only the object $p1$ satisfies the last predicate and it is included in the answer set.

Let us consider the virtual class (v-3) given in Section 3.2.4 as another example. In this case, the virtual class inherits from two classes, *pollutants* and *sites*. Assume that the following query is activated:

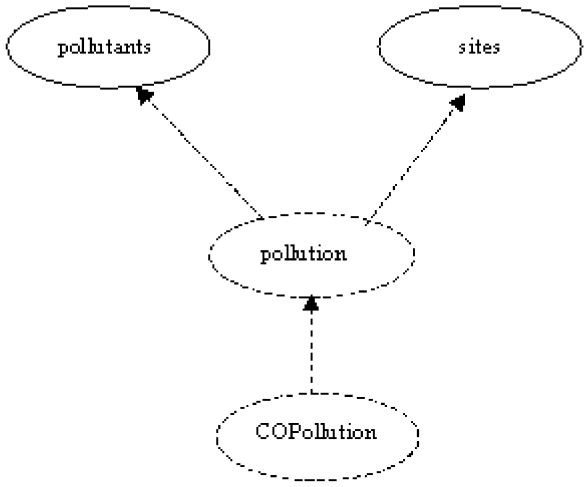

Fig. 3. Virtual class inheritance.

select *X* ***from*** *pollution(X)*
where *X.status([veryDangerous], 0.8);*

- The bridge identifies *pollution* as a virtual class defined on the FKB.
- It transfers the objects of the *pollutants* and *sites* classes from the FOOD system to the FKB memory. Since there is no specified object threshold level in the virtual class definition, the threshold value is assumed as one (1) by default and the objects having membership degree one (1) to the *pollutants* and *sites* classes are transferred.
- New objects are derived evaluating the condition statement on the both the *pollutants* and the *sites* objects.
- At the last step, the $X.status([veryDangerous], 0.8)$ predicate is evaluated and the satisfying objects are sent to the user interface.

It is possible to define a virtual class inherited from previously defined virtual classes. For example, consider the class schema given in Fig. 3. In this schema, *pollutants* and *sites* are normal database classes. *Pollution* and *COPollution* are virtual classes and *COPollution* inherits from *pollution*, which is also a virtual class. Notice that *contaminant* is an attribute of *pollution* and *contaminants* is another class defined to handle specific information to contaminants causing pollution.

vclass *COPollution;*
inherits *pollution;*
condition *COPollution(X):-*
pollution(X,0.7), X.contaminant(co),
X.status([veryDangerous],0.8),
contaminants(co),
co.healthEffect([extremelyBad],0.8);
endclass;

Now, we can formulate a query using *COPollution*.

select X.location **from** COPollution(X,0.8);

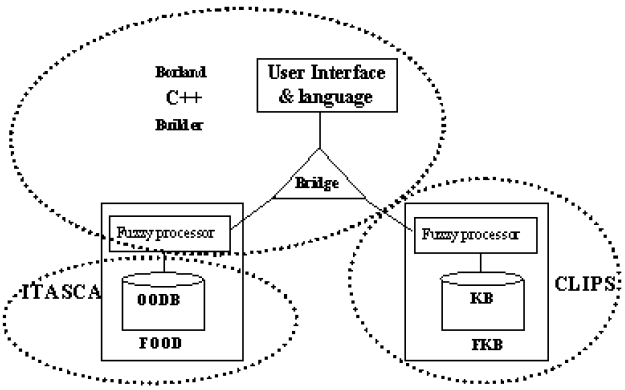

Fig. 4. The implementation platform of the prototype system.

- The bridge identifies *COpollution* as a virtual class from its catalogue. Moreover, its superclass is also a virtual class.
- Therefore, the objects of the *pollution* virtual class are derived as explained in the previous example. Then, the condition of *COpollution* is applied and its objects are derived at the second step.
- At the last step, the satisfying objects are sent to the user interface.

6 Implementation of the IFOOD Architecture

We implemented a prototype system of the IFOOD architecture. C Language Integrated Production System (CLIPS) is utilized for implementation of the knowledge base component and ITASCA is used for implementation of the object-oriented database component. The reasons to select CLIPS as the tool for developing FKB are as follows:

1. It is a powerful tool to develop expert systems.
2. It supports object-oriented programming.
3. It is designed for full integration with other languages and it is possible to integrate it through the C++ API's.
4. It is available on both Unix and PC Windows environment.
5. It is free and its source code is available for further developments.

We used C++ programming language for the implementation of the user interface, the bridge, and partially the fuzzy processors. The overall implementation platform is shown in Fig. 4.

CLIPS is extended in a way to support fuzzy inferencing. Pattern matching of CLIPS is already working for crisp attributes. However, when there is a fuzzy attribute in a rule definition, the new fuzzy pattern matching function is needed. Therefore, we developed a fuzzy pattern-matching module. The fuzzy pattern matching module returns a real value in [0,1] that shows the membership degree of object attribute to the rule antecedent. Then, the inference engine compares this value with the specified threshold. If the value is greater than or equal to the threshold, then rule evaluation continues with the next antecedent clause, otherwise the rule fails. If all the antecedent clauses are

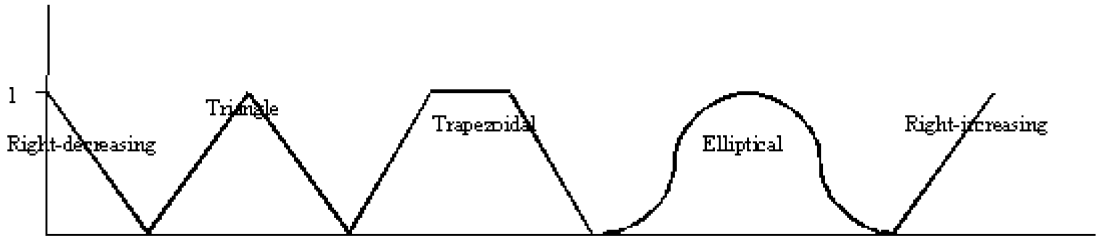

Fig. 5. Membership function supported by the IFOOD system.

succeeded, then the membership degree of the rule conclusion is calculated.

CLIPS supports pattern matching on objects too. The class schema must be defined before the definition of rules on objects. Therefore, the class schema defined in the IFOOD model is redefined on both ITASCA and CLIPS as a mirror of each other. The rules are inserted into CLIPS after class definitions and the objects are inserted into ITASCA. When there is a rule firing on CLIPS, the required objects are retrieved from ITASCA and inserted into CLIPS in the appropriate format as the instances of the priori-defined classes. Then, the fired rule is evaluated on the transferred objects.

The Borland C++ Builder is used to implement the user interface of the system. A graphical user interface is constructed to facilitate the usage of the system. The user is able to create, modify, and delete fuzzy types, similarity matrices, membership functions, classes, objects, and rules through the user interface. The user is also able to form queries to retrieve both information stored on the database and the new information deduced from the fuzzy knowledge base.

The ITASCA Object Database Management System is integrated into the architecture by using the C++ API library. The application can run on any machine in the network. Access to the database is available regardless of the machine on which the application runs. A front-end processor is developed to provide the fuzzy processor functionality shown in Fig. 4 on top of the object-oriented database. This front-end processor maps requests from the bridge or the user interface to the appropriate format for the object-oriented database and it submits the data coming from the object-oriented database to the upper layer. Therefore, the fuzzy processor is a mapping tool between a crisp world and a fuzzy world.

The catalog is implemented under the IFOOD system to keep necessary information about the application. This information is about the classes, the fuzzy attributes, the fuzzy attribute templates, the fuzzy attribute relevance values, etc. Additionally, information about virtual (derived) attributes and classes is also kept in the catalog for further processing. When the user activates a query, the system evaluates it according to information stored in the catalog.

The functions of the application are grouped into six modules at the user-interface level. The first module is for the fuzzy definitions. In this module, the user defines fuzzy types, similarity relations, and membership functions. Five predefined membership functions shown in Fig. 5 are available in the system.

The second module provides the ability to create a new application, define the class schema, and update the schema. When a user creates a new application, the application database and the IFOOD catalog are constructed. Users can define classes including rule definitions at the user-interface level. The attributes defined can be of a fuzzy type and the derived attributes can be defined using fuzzy attributes. The fuzzy types have to be defined before using them in class definitions. The class definition is processed at four steps by the bridge:

- The class definition is inserted into ITASCA ODBMS at the first step.
- The same definition is created on CLIPS FKB at the second step.
- If it includes any derived attribute, the rule is inserted into CLIPS FKB at the third step.
- Finally, the required information is stored in the application catalog. If an error occurs at any step, a rollback action is applied and the user is informed about the error.

The third module is for object manipulation. The user can create, update, and delete objects of any class. The membership degree of an object is calculated automatically and stored in the database. The membership degree of an object is one, if the class is crisp; otherwise, it is between zero and one. This module is directly related to ITASCA since objects are stored in the object-oriented database.

The fourth module is implemented for derived attribute operations. Although there is a possibility of defining derived attributes in class definitions, we developed this module to manipulate derived attributes more efficiently. In this module, users can define new derived attributes or delete the derived attributes connected to one of the classes in the class schema.

The fifth module includes in the derived (virtual) class operations. In the prototype, users can create a derived class, which is a specialization of the database classes. The derived class definition is inserted into the CLIPS class schema and required information about derived class is stored in the IFOOD catalog. Derived classes are totally handled by CLIPS FKB. The defined classes are kept on CLIPS FKB until the user deletes them explicitly. Their instances are derived from the object-oriented database according to the definitions.

The last module that we should mention here is the query module. The system provides three types of query facility. The first query is the class browser. Users are able to

access the objects of any class in the class hierarchy in the graphical user interface. They can also define filters to set criteria's to select specific objects. The second query possibility is the object browser that provides access to a specific object in the graphical user interface. The third query option is the *select-from-where* query structure as explained in Section 3.2.6. Users can formulate queries with crisp and fuzzy conditions. A query can be related to both data stored in ITASCA OODB and rules (derived attributes and classes) in CLIPS FKB. This type of query is evaluated at three steps by the bridge mechanism:

- At the first step, the crisp data is retrieved from ITASCA ODBMS according to the crisp specifications in the query.
- At the second step, the fuzzy processor evaluates the fuzzy criteria. If there are no fuzzy criteria specified in the query, this step is skipped.
- At the third step, the selected objects are transferred to the CLIPS FKB memory and the related rules are fired. If there is no specification related to CLIPS FKB in the query, this step is skipped. The result of the query is submitted to the user interface after the query evaluation is done.

7 Conclusions

The main motivation of this study is to develop a powerful and intelligent database environment for knowledge intensive applications. In this research, we introduced an integrated architecture for an intelligent object-oriented database system that allows users to specify their applications with uncertain and fuzzy properties, such as membership functions, similarity relations, and uncertain types. The objects/classes defined in the environment may have uncertain attributes and rules. Virtual classes are created to provide different views of the class schema. Objects can have uncertain and fuzzy characteristics. Users can query the data/knowledge base that may include uncertainty by specifying either crisp or fuzzy queries. At the lower level of the architecture, the fuzzy object-oriented database system handles the large scale of complex and uncertain data and the fuzzy knowledge base handles knowledge definitions and fuzzy deduction. These two systems are coupled through a bridge mechanism that provides interoperability. Translation between the two components through the bridge mechanism may result in some inefficiency for the architecture. One solution to this is to develop a fuzzy deductive object-oriented database model for a fully integrated environment. The development of a fuzzy object-oriented database model is still a research issue and it requires more research effort.

A prototype of the system is implemented by utilizing ITASCA Object Database Management System, CLIPS, and C++ programming languages. The implementation of the prototype system illustrates the feasibility of the proposed IFOOD architecture for the next generation information systems. As it is the case for all the knowledge-based systems, when the number of rules in the FKB increases, the efficiency of the system decreases. Therefore, the scalability of the architecture is limited by the number of rules existing in the FKB. Our research effort for improving various aspects of the architecture (i.e., increasing the scalability of the system) and developing a more efficient fuzzy inference mechanism are ongoing.

References

[1] J.F. Baldwin and T.P. Martin, "Fuzzy Objects and Multiple Inheritance in Frill++," *Proc. EUFIT '96*, pp. 680-684, 1996.

[2] F. Bancilhon, C. Delobel, and P. Kanellakis, *Building an Object-Oriented Database System.* Morgan Kaufmann Publishers, Inc., 1992.

[3] M.L. Barja, A.A. Fernandes, N.W. Paton, M.H. Willams, A. Dinn, and A.I. Abdelmoty, "Design and Implementation of Rock & Roll: A Deductive Object Oriented Database System," *Information Systems,* vol. 20, no. 3, pp. 185-211, 1995.

[4] P. Bosc and O. Pivert, "SQLf: A Relational Database Language for Fuzzy Quering," *IEEE Trans. Fuzzy Systems,* vol. 3, no. 1, pp. 1-17, 1995.

[5] K.S. Candan and W. Li, "On Similarity Measures for Multimedia Database Applications," *Knowledge and Information Systems,* vol. 3, pp. 30-51, 2001.

[6] R.M. Colomb, *Deductive Databases and Their Applications.* Taylor & Francis, 1998.

[7] M. Dahr, *Deductive Databases: Theory and Applications.* Int'l Thomson Publishing Services Ltd., 1996.

[8] D. Dubois, H. Prade, and J. Rossazza, "Vagueness, Typicality, and Uncertainty in Class Hierarchies," *Int'l J. Intelligent Systems,* vol. 6, pp. 167-183, 1991.

[9] R. Elmasri and S.B. Navathe, *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Co., 2000.

[10] R. George, R. Srikanth, F.E. Petry, and B.P. Buckles, "Uncertainty Management Issues in the Object-Oriented Data Model," *IEEE Trans. Fuzzy Systems,* vol. 4, no. 2, pp. 179-192, 1996.

[11] I. Graham, "Fuzzy Logic in Commercial Expert Systems—Result and Prospects," *Fuzzy Sets and Systems,* vol. 40, pp. 451-472, 1991.

[12] N.V. Gysehem, R. De Caluwe, and R. Vandenberghe, "UFO: Uncertainty and Fuzziness in an Object-Oriented Model," *Proc. FUZZ-IEEE '93,* 1993.

[13] J.L. Harrington, *Object-Oriented Database Design.* Morgan Kaufmann Publishers, Inc., 2000.

[14] Y. Inoue, S. Yamamoto, and S. Yasunobu, "Fuzzy Set Object: Fuzzy Set as First-Class Object," *Proc. IFSA '91 Conf.,* 1991.

[15] L. Kershberg, *Expert Database Systems.* Benjamin/Cummings, 1989.

[16] M. Kifer, G. Lausen, and J. Wu, "Logical Foundations of Object-Oriented and Frame-Based Languages," Technical Report 90/14, Dept. of Computer Science, State Univ. of New York at Stony Brook (SUNY), June 1990.

[17] *Object-Oriented Concepts, Databases and Applications.* W. Kim and F.H. Lochovsky, eds., Addison-Wesley, 1989.

[18] M. Koyuncu, A. Yazici, and R. George, "IFOOD: An Intelligent Object-Oriented Database Architecture," *Proc. 10th Int'l Conf. Database and Expert System Applications (DEXA '99),* pp. 36-45, Aug. 1999.

[19] M. Koyuncu, A. Yazici, and R. George, "Flexible Querying in an Intelligent Object-Oriented Database Environment," *Proc. Fourth Int'l Conf. Flexible Query Answering Systems (FQAS '2000),* pp. 75-84, Oct. 2000.

[20] S. Krishna, *Introduction to Database and Knowledge-Based Systems.* World Scientific, 1992.

[21] J. Lee, N. Xue, K. Hsu, and S.J. Yang, "Modeling Imprecise Requirements with Fuzzy Objects," *Information Sciences,* vol. 118, pp. 101-119, 1999.

[22] Q. Li and F.H. Lochovsky, "ADOME: An Advanced Object Modeling Environment," *IEEE Trans. Knowledge and Data Eng.,* vol. 10, no. 2, pp. 255-276, 1998.

[23] W. Lipski, "On Semantic Issues Connected with Incomplete Information," *ACM Trans. Database Systems,* vol. 4, no. 3, pp. 262-296, Sept. 1979.

[24] M. Liu, "An Overview of the Rule-Based Object Language," *J. Intelligent Information Systems,* vol. 10, pp. 5-29, 1998.

[25] M. Liu, "Deductive Database Languages: Problems and Solutions," *ACM Computing Surveys,* vol. 31, no. 1, pp. 27-62, 1999.

[26] F.G. McCabe, *Logic and Objects.* Prentice Hall, 1992.
[27] A. Motro, "Accommodating Imprecision in Database Systems: Issues and Solutions," *SIGMOD Record,* vol. 19, no. 4, pp. 69-74, Dec. 1990.
[28] T.D. Ndousse, "Intelligent Systems Modeling with Reusable Fuzzy Objects," *Int'l J. Intelligent Systems,* vol. 12, pp. 137-152, 1997.
[29] S. Parsons, "Current Approaches to handling Imperfect Information in Data and Knowledge Bases," *IEEE Trans. Knowledge and Data Eng.,* vol. 8, no. 3, pp. 353-372, June 1996.
[30] F.E. Petry, *Fuzzy Databases, Principles and Applications.* Kluwer Academic Publishers, 1996.
[31] Y. Takahashi, "Fuzzy Database Query Languages and Their Relational Completeness Theorem," *IEEE Trans. Knowledge and Data Eng.,* vol. 5, no. 1, pp. 122-125, 1993.
[32] M. Umano, T. Imada, I. Hatono, and H. Tamura, "Fuzzy Object-Oriented Databases and Implementation of its SQL-Type Data Manipulation Language," *Proc. IEEE Int'l Conf. Fuzzy Systems,* vol. 2, pp. 1344-1349, 1998.
[33] S.D. Urban, A.P. Karadimce, S.W. Dietrich, T.B. Abdellatif, and H.W. Rene Chan, "CDOL: A Comprehensive Declarative Object Language," *Data & Knowledge Eng.,* vol. 22, pp. 67-111, 1997.
[34] T. Warren, L.Z. Zhang, and C. Mount, "Similarity Measures for Retrieval in Case-Based Reasoning Systems," *Applied Artificial Intelligence,* vol. 12, no. 4, pp. 267-288, 1998.
[35] A. Yazici, R. George, and D. Aksoy, "Design and Implementation Issues in the Fuzzy Object-Oriented Data (FOOD) Model," *Information Sciences (Int'l J.),* vol. 108, no. 4, pp. 241-260, 1998.
[36] A. Yazici and R. George, *Fuzzy Database Modeling.* New York: Physica-Verlag, 1999.
[37] A. Yazici, A. Soysal, B.P. Buckles, and F.E. Petry, "Uncertainty in a Nested Relational Database Model," *Data & Knowledge Eng.,* vol. 30, pp. 275-301, 1999.
[38] A. Yazici, B.P. Buckles, and F.E. Petry, "Handling Complex and Uncertain Information in the ExIFO and NF2 Data Models," *IEEE Trans. Fuzzy Systems,* vol. 7, no. 6, pp. 659-676, Dec. 1999.
[39] J. Yen and R. Langari, *Fuzzy Logic, Intelligence, Control, and Information.* Prentice Hall, 1999.
[40] L.A. Zadeh, "Similarity Relations and Fuzzy Orderings," *Information Sciences,* vol. 3, no. 2, pp. 177-200, 1971.
[41] M. Zemankova-Leech and A. Kandel, *Fuzzy Relational Databases—A Key to Expert Systems.* Verlag, 1984.

Murat Koyuncu received the BS degree in electronics in 1986, the MS degree in computer engineering in 1995, and the PhD degree in computer engineering in 2001 from Middle East Technical University, Ankara, Turkey. His research interests include fuzzy logic, object-oriented databases, and knowledge-based systems.

Adnan Yazici received the PhD degree in 1991 from the Department of Computer Science at Tulane University, New Orleans. He is currently a professor in the Department of Computer Engineering at Middle East Technical University, Ankara, Turkey. His current research interests include intelligent database systems, fuzzy database modeling, design and analysis of algorithms, and spatio-temporal databases. Professor Yazici has published more than 60 international technical papers and coauthored a book entitled *Fuzzy Database Modeling* (Springer-Verlag, 1999). He is a senior member of IEEE. He serves as a committee member and reviewer of some international journals and conferences.