Conceptual Modeling of Geographic Information System Applications

Adnan YAZICI[1] and Kemal AKKAYA[1]

[1] Department of Computer Engineering, Middle East Technical University, 06531, Ankara / Turkey, e-mail: {yazici,akkaya}@ceng.metu.edu.tr

Abstract. An important research trend in databases is to handle different types of uncertainty at both conceptual and logical levels for various non-traditional applications that may involve imprecision and uncertainty that have been difficult to integrate cohesively in simple database models. In this study we describe how to conceptually model complex and uncertain information at the conceptual level (by utilizing the ExIFO$_2$ model) for Geographic Information System (GIS) applications. We also give the mapping algorithm that transforms the conceptual schema of GIS applications into the logical database structures by utilizing the fuzzy object-oriented database (FOOD) model at the logical level. The types of uncertainty that we mainly focus in this paper are fuzzy, null, and incomplete information related to objects and their properties, classes, and relationships among objects and classes of GIS applications.

Keywords. Uncertainty, Fuzziness, Object-Oriented Databases, Conceptual Modeling, and Geographic Applications

1 Introduction

In general, a conceptual model is a type of abstraction that uses logical concepts and hides the details of implementation and data storage. Some of the existing conceptual models offer powerful concepts to the designers that provide getting the most complete specification from the real world [1,4,13,15]. However, the classical models in this group generally suffer from the lack of concepts (object-identity, reusability, representation of imprecise information, etc.) which are important for modeling various complex applications such as Multimedia Database Systems (MMDBS), Office Automation Systems (OAS), Geographic Information Systems (GIS), Expert Database Systems (EDS) and so on. Modeling requirements for these complex applications and flaws in the currently used models have led to the definition of more powerful conceptual models such as the ExIFO$_2$ model [17,18] and the others [5]. However, there still have been a number of research issues related to conceptual modeling with increasing complexity in the complex applications. As some researchers [2,4,12,15] have pointed out, the general-purpose conceptual models are not adequate for GIS applications. Therefore, there is a need for utilizing a powerful conceptual model to satisfy the unique requirements of these applications, such as the specific properties of geographic objects or entities and relationships, which may involve various forms of uncertainty.

There are two common data models for modeling spatial information in GIS applications: field-based models and object-based models [11]. Field-based models treat the spatial information space as a continuous domain such as altitude, rainfall and temperature - as a collection of spatial functions transforming a space-partition to an attribute domain. The object-based models treat the information space as if it is populated with recognizable objects (spatial and aspatial) that are discrete and spatially referenced. The modeling approach that we take in this study is object-based. That is, the database for GIS applications stores a map that consists of a collection of identifiable objects, which refer to the partitions and fragments of information space. Geographical objects have geometric properties that can be modeled by the measurements, properties, and relationships of points, lines, and polygons. Some geographic objects may also have topological properties.

There have been previous works related to conceptual modeling of geographic applications by either introducing a new conceptual model or extending one of the existing models, mainly the Enhanced Entity-Relationship (EER) model. GraphDB [3], GODOT [2], and the system by Worboy [14] are some of those which attempt to model GIS applications using an object-based approach. GISER [12] and Geo-ER [4], which are extensions to the ER model, are some of recent attempts that try to unify continuous fields with entities and relationships. The Geo-ER model provides a set of concepts as an add-on to the ER model and attempts to capture spatial peculiarities at the conceptual level of geographic database design. The GISER model is based on four major concepts. These concepts are: space/time (boundless multidimensional extends in which geographic phenomena and events can occur and have relative position and direction); feature (geographic phenomena such as cities, mountains, etc.); coverage (a set of spatial objects); and spatial object (a subset of space and time.) All of these conceptual models assume that GIS applications do not involve any form of uncertainty.

However, it is not always possible to describe all the semantics of real world information precisely since the observation and capturing of some real world objects are not perfect, hence its modeling and representation are deficient. Uncertainty might arise from the data itself and/or the relations between objects. As a consequence of this, queries involving imprecise and uncertain information may be unavoidable. The management of uncertainty has been ignored for a long time, but it is now inevitable to be able to incorporate uncertainty in many complex applications including GIS applications. More specifically, the need for incorporating uncertainty in a conceptual modeling of GIS applications arises from the following reasons: (1) Some geographic information in GIS applications is inherently imprecise or fuzzy. For example, locations of geographic objects, some of spatial relationships, and various geometric and topological properties usually involve various forms of uncertainty. (2) Only few natural geographic phenomena have boundaries that can be accurately represented as mathematical lines. For example, vegetation types, soils, slopes, wildlife habitats, etc., all have fuzzy boundaries due to the transitional nature of variation in the phenomenon. (3) Some measurements related to spatial domain are often incomplete. Sometimes forcing such data to be completely crisp may result in falsity and useless information. (4) In GIS applications, some of the spatial domain related knowledge is specified in natural language by using fuzzy terms and numerous quantifiers (e.g., many, few, some, almost, etc.). Most of these quantifiers are fuzzy and are used when conveying vague information.

Since almost none of the published studies modeling GIS applications deals with uncertain information at the conceptual level, there is a need not only for dealing with spatial objects, but also for handling uncertain properties including inherently fuzzy ones. In addition, transforming the conceptual schema into the logical one (the object-oriented database (FOOD) model is utilized in this study) should be done in a way that it is straightforward and preserves most information.

In this paper a conceptual model, called the ExIFO$_2$ model, is utilized to conceptually model geographic applications. This model includes representation of spatial objects along with their geometric and/or topological properties in addition to uncertain ones. The conceptual ExIFO$_2$ model attempts to preserve the acquired strengths of semantic approaches, while integrating concepts of the object-oriented paradigm and uncertainty.

The next section includes a brief description of the ExIFO$_2$ model [17,18] and the fuzzy object-oriented database (FOOD) model [16,17]. In Section 3 we present conceptual modeling of GIS applications. Before the conclusions in the last section, we present the mapping algorithm that transforms the conceptual schema represented with the ExIFO$_2$ model into a logical database description by utilizing the fuzzy object-oriented database (FOOD) model.

2 Background

In this section, we discuss the basic definitions and characteristics of the models and concepts used. Topics include a brief background on the ExIFO$_2$ model and the FOOD database model.

2.1 The ExIFO$_2$ Model

The ExIFO$_2$ model is an extension of the combination of the semantic data models IFO$_2$[9] and ExIFO[17,18], which are each an extension of the IFO model [1]. The ExIFO$_2$ model is a mathematically defined data model that incorporates the fundamental principles of semantic database modeling within a graph-based representational framework. More formally, an ExIFO$_2$ schema is a directed graph with various types of vertices and edges, representing atomic objects, constructed objects, fragments and ISA relationships.

Two types of uncertainty are dealt with in the ExIFO$_2$ model. The first type of uncertainty is considered to be at the *attribute level*, meaning that some attributes of objects may have uncertain values. The second type of uncertainty is considered to be at the *object* and *class level*. That is, some objects may have instances whose membership to the object set may be graded in [0,1] (i.e., partial membership of an instance of an object type) and/or some subclasses whose membership to the superclasses (class/subclass level uncertainty) may be in [0,1]. An explicit definition of the object identifier, which is object value independent, is integrated in the model. To achieve these, all manipulated elements of the ExIFO$_2$ model are redefined. On the other hand, integrating the concepts of *alternative*, *composition* and *grouping* for complex objects enhances the modeling power of the ExIFO_2 model.

Graphical representation and a brief description of the related concepts of the ExIFO_2 model are included below. All types of the ExIFO_2 model and constructors are shown graphically in Figure 1, Figure 2 and Figure 3.

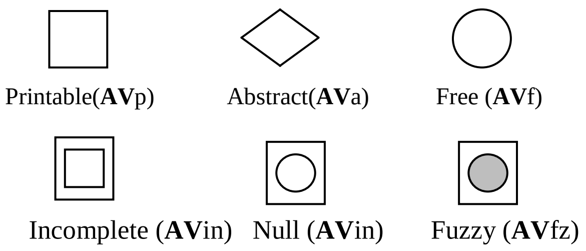

Fig. 1. Fuzzy, Incomplete, Null, and Atomic Types

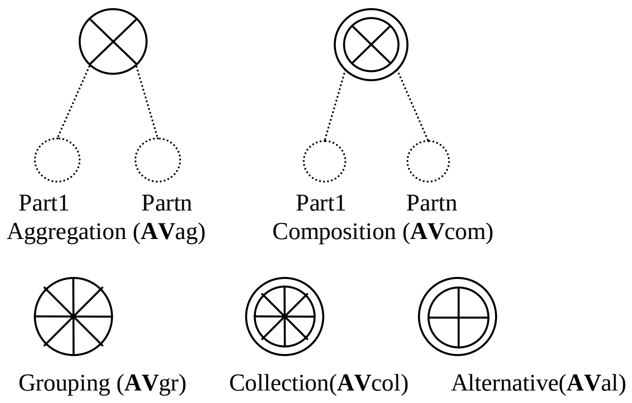

Fig. 2. Complex Types

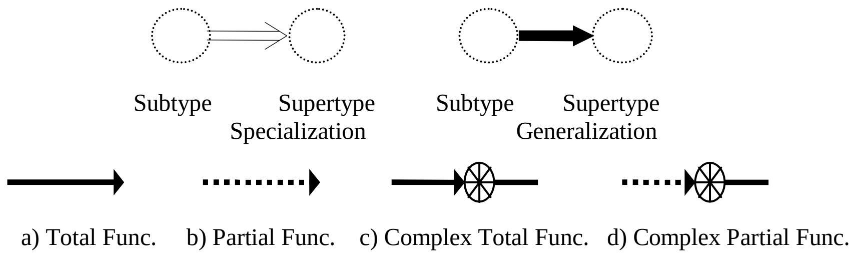

Fig. 3. Function Types and ISA Links

We consider three kinds of uncertainty in the ExIFO_2 model:

The true data may belong to a specific set of values, *incompleteness (or imprecision)*, The true data value is not known, *null*,

The true data is available, but specified only with a descriptive term, *fuzzy*.

Uncertainty is also considered at two levels: *Attribute* level and *object-class* level. For the representation of attribute level uncertainty in the ExIFO$_2$ model, three constructors (*fuzzyset*, *incompleteset*, and *null* constructors) are defined. Using these constructors it is possible to explicitly represent attributes having uncertain values. The second level of uncertainty considered is the fuzziness of instances of specific objects in the corresponding object set (class/object level) and fuzziness of subclasses in superclasses (class/subclass level). In class/subclass level "F" will be used on generalization and specialization arrows in the representation of the object, to indicate the possibility of gradual membership. (e.g. a *caravan* class can be considered as a subclass of *house* class with a membership degree of 0.6).

There are two main groups of types in the ExIFO$_2$ model. The first group are the atomic types (*printable*, *abstract* and *free*) and the second group are the complex types (*aggregation*, *composition*, *collection*, *grouping*, *union*, *fuzzySet, incomplete, and null* costructors) produced by the use of constructors.

The *printable* objects correspond to objects of predefined types that serve as the basis for input and output. The *abstract* objects correspond to objects in the world that have no underlying structure. The third type of atomic objects is *free*, and corresponds to entities obtained via the *'ISA'* relationship. The graphical representations of these atomic types are given in Figure 3. Non-atomic (or constructed) types are formed from an underlying type by applying a finite set of constructors. These constructors (except *fuzzyset, incompleteset* and *null* constructors) may be applied recursively to build more complex types.

The *grouping* constructor is a high-level mechanism of the ExIFO$_2$ model and is an abstraction in which the relationship among elements is considered as a set of objects. This constructor depicts the type corresponding to sets of data values of an attribute. The *grouping* constructor has 'AND' semantics, since each member of the set necessarily and precisely belongs to the set. Hence, this type of the constructors can capture the attributes that are both multi-valued and crisp. All objects in the *grouping* abstraction are of the same type and not ordered. An example for this constructor is the set of *authors* of a book. A set of authors of a specific book is precisely known and crisp; therefore, the authors are related with '*AND*' semantics. That is, the names of the authors specified are the only authors of the book, not some subset of the names of that set.

The *collection* types include an exclusivity constraint for the *grouping* constructor and this is the only difference between the *collection* and *grouping* constructors.

The *aggregation* constructor represents the *aggregation* abstraction of semantic models defined by the Cartesian product. This constructor connects a subtype representing a part of an object to the type representing the entire object; thus, building a higher level object. This abstraction ignores some individual differences of the aggregated types. For example, object type *motor-boat* is viewed as being an ordered pair of *hull* and *motor*. Semantically, the *aggregation* constructor derives only objects fully supported by the type, but, unlike the *grouping* constructor, the objects are ordered. An example is the list of components of an address of a person (i.e., *street-name, zip code, city*, etc.).

The *composition* constructor also represents the *aggregation* abstraction of semantic models defined by the Cartesian product. The only difference of this constructor from the *aggregation* constructor is that it provides an exclusive structure.

In the ExIFO$_2$ model, structurally different types are handled by using the *alternative* type concept. This constructor represents the IS_A generalization link enhanced with a disjunction constraint between the generalized types.

With a *fuzzyset* constructor, a set of values of a given specified domain of a type (attribute in relational model terminology) can be defined. Only a subset of these values that corresponds to the object of that structure type are true instances. This constructor builds an instance in the form of a fuzzy set whose elements are related to each other fuzzily. That is, any instance of the set belongs to the set with a degree in [0,1] and all of the elements are said to have similarity relationship to each other to that degree[18].

The *incompleteset* constructor is used to represent *incomplete* information, such as range values. Semantically, an object is the only instance of the corresponding type. This constructor represents a given specified domain of a type, which the constructor defines, but it involves different semantics compared to the *fuzzyset* constructor. Only one of the values from a specified range corresponds to an instance of that structure type. Therefore, the *incompleteset* constructor has ‘XOR’ semantics, since only one instance of the set represented as a range is the true instance of the underlying type.

Another kind of constructor for representing uncertainty is the *null* constructor. This constructor is used to represent attributes whose domain is extended to handle various types of null values. The null values taken into consideration in the ExIFO$_2$ model are as follows: the true value is not known (*unk*), does not exist (*dne*), or there is no information (*ni*) on whether a value exists or not. All of these can be considered as a type of a null value and depending on the situation, different interpretations given to the null value. For example, if we know that a person does not have a telephone, then it can be evaluated as *dne*. If we know the person has a telephone but we do not know the number, then it is *unk.* If we do not know whether or not a telephone exists, then it is *ni*.

The types in the ExIFO$_2$ model can be linked by using functions (called *fragments*). The goal of a fragment is to describe the properties of the principal types. Functions can be (simple, complex -multivalued-) or (partial-0:N link-, total-1:N link-). In a partial function some elements of the domain have no associated elements in the codomain, otherwise it is called a total function [3]. Any total or partial function can be complex or simple.

A set of schema invariants and update rules that consists of the explicit statement of some rules and constraints implicit in the model are defined. Additionally, some of the restrictions concerning the fuzzy and object-oriented extensions are also included in the model. In order to provide a verification of an ExIFO$_2$ schema while preserving a consistent representation, a set of update rules are defined. These rules are used especially for modifying an existing ExIFO$_2$ schema. Whenever a change is done to the schema, these rules are checked to see if the schema invariant constraints are violated or not. Here we do not describe all of the details of the ExIFO$_2$ model due to space limitations. The details (along with a number of examples) of the model can be found in [17,18].

2.2 The Fuzzy Object-Oriented Data (FOOD) Model

Here we briefly review the fuzzy object-oriented data model (the FOOD model) [16,17], since the FOOD model is utilized as the logical model into which to map the conceptual schema of a GIS application. The basis of the fuzzy object-oriented database model is the similarity relation. For each fuzzy attribute, a fuzzy domain and a similarity matrix are defined. Similarity matrices are used to represent the relation within the fuzzy attributes. The domain, *dom*, is the set of values the attribute may take, irrespective of the class it falls into. The range of an attribute, *rng*, is the set of allowed values that a member of a class, i.e. an object, may take for an attribute. In general $rng \subseteq dom$. For instance, assume that *height* is an attribute and the domain of *height* is between 0 and 230 cm. If there exists a *Student* class, the range of the *height* attribute for this class may be 130cm to 230cm. 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 $a_i \in Attr(C) = \{a_1, a_2, ..., a_n\}$. $Attr(C)$ refers to the attributes of class C. Similar objects are grouped together to form a class and fuzziness at object/class and class/superclass levels are represented this way. The idea of fuzziness extends in the relation of an object with the class of which is created as an instance. An object belongs to a class with a degree of membership. Based on the considerations of relevance and ranges of attribute values, the membership of object o_j in class C with attributes $Attr(C)$ can be defined as

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

where $RLV(a_i,C)$ indicates the relevance of attribute a_i to class C, and $INC(rng_C(a_i)/o_j(a_i))$ denotes the degree of inclusion of the attribute values of o_j in the formal range of a_i in class C. The degree of inclusion, determines the extent of similarity between a value (or a set of values) in the denominator with a value (or a set of values) in the numerator. Function f represents the aggregation over n attributes in the class and g reflects the type of link existing between an instance (object) and a class/superclass (f and g are functions that may be inherited from the superclass or may be defined within the local class). The value of $RLV(a_i,C)$ may be supplied by the user or computed. Several cases (which we will not discuss here) are possible for the evaluation of $INC(rng(a_i)/o_j(a_i))$ [16].

Fuzziness may occur at three different levels in our fuzzy object-oriented database model; the attribute level, the object/class level and the class / superclass level. We will only summarize some of the basic concepts of the model here, a detailed description along with the examples of how fuzziness is handled in each level in the model is given in [16,17].

Attribute Definitions

Unlike the fuzzy relational model, in the fuzzy object-oriented model, attributes can have a set of values (leading to multivalued attributes) connected with a logical operator AND/OR/XOR. The attribute value sets are differentiated according to their semantics. The following notation is used to indicate AND, OR or XOR multivalued attributes:

logical operator: AND:<.......>, OR:{..... } and XOR:[......]

Assume that the domain of foreign languages is = *{English, German, French, Italian, Spanish, Dutch, Turkish, Chinese, Japanese, Russian}*. The following interpretations are valid:

Mathew.Lang = <English, German>,
i.e., "Mathew can speak English and German".
Helga.Lang = {German, French},
i.e., "Helga can speak German or French or maybe both."
Hazal.Lang = [Italian, Turkish],
i.e., "Hazal can speak one language either Italian or Turkish.".

Class Definitions

Every class has a range definition for each of the fuzzy attributes with the corresponding relevance rules indicating the importance of that attribute in the definition of that class. In this way an "approximate" description of the class is given. An attribute of a class is allowed to take any value from the domain without considering the range values.

In the FOOD model, semantics is associated with the range definitions to permit a more precise definition of a class. The set given in the similarity-based range definition includes the semantics *OR*, *AND*, or *XOR*. *AND* semantics forces a multivalued use of the attribute and *XOR* forces a singleton attribute. *OR* is the most uncertain definition that can be made for a class definition. The logical relation is determined in the range definition, and the instances with multivalued attributes obey this relation. For instance class *C* having attributes *a, b, c* from domains *A*, *B*, and *C* respectively can be defined as:

Class C:

$rng_C(a) = \{a_1, a_3, a_6\}$ $\quad dom_C(a) = \{a_1, a_2, a_3, ..., a_k\}$
$rng_C(b) = <b_5, b_7>$ $\quad dom_C(b) = \{b_1, b_2, b_3, .., b_m\}$
$rng_C(c) = [c_2, c_4, c_6]$ $\quad dom_C(c) = \{c_1, c_2, c_3, ..., c_n\}$

Class *C* has "a_1, a_3 or a_6" values for attribute *a*, "b_5 and b_7" requiring multivalued use of attribute *b*, and finally a value of either "c_2", "c_4", or "c_6" for 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.

If the range definition of attribute *age* of class *Person* was given *as rng(age) = {young, very young}*, then the objects use the same logical operator (*OR*) for attribute *age*:

$o_1(age) = \{very\ old, old\}$, $o_2(age) = \{very\ old, young, old\}$

Relevance weights are assigned for each attribute, and they show the significance of the range definition of that attribute on the class definition. If relevance rules for class *C* are given as:

$RLV(a) = 1.5, RLV(b) = 2.5, RLV(c) = 0.2$

Then attribute *b* is the most important attribute defining class *C*, attribute *a* more or less determines class *C*, and attribute *c* is of very little importance in determining class *C*.

Object/ Class Relations

The object/class level denotes the membership degree of an object to a class. The main feature that distinguishes fuzzy classes from crisp classes is that the boundaries of fuzzy classes are imprecise. The imprecision of the attribute values causes imprecision in the class boundaries. Some objects are full members of a fuzzy class with membership degree 1, but some objects may be related to this class with a degree between 0 and 1. In this case they 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, the membership degree of an object to the class is calculated 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 the determination of the boundary of a fuzzy class. If an object has the ideal values for each fuzzy attribute, then this object is an instance of that class with a membership degree of 1. Otherwise, it is either an instance with a membership degree less than 1, or it is not an instance at all (when the membership degree is smaller than the threshold value) depending on the similarities between attribute values and formal range values. The closer the attribute values to the range, the higher the membership degree of the object. If an attribute value of an object is crisp, the membership degree of this crisp value to the fuzzy terms in formal range definition is calculated and used to find the object membership degree to the class. The system calculates the membership degree of objects to their classes during object creation and updating by using the formulas given below with the related semantics.

To calculate the membership degree of an object to a class, we 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 value to the range definitions, the higher the class/object membership degree. But how is this inclusion determined? The membership degree of object o_j to class C is determined using:

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

where $INC(rng_C(a_i)/o_j(a_i))$ 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 class C and is given in the class definition. The weighted-average is used to calculate the membership degree of objects. All attributes, therefore, affect the membership degree proportional to their relevance.

The formulas used to calculate inclusion degrees for the *AND*, *OR*, and *XOR* connection semantics are given in [16,17].

Class/Subclass Relation

The next importance relationship is between a class and superclass, in other words the answer to the question "To what extent the class belongs to its superclass". The FOOD model is the basis of the formulation to find the degree of membership of a class in its superclass.

In the FOOD model, an approximation approach is used to find the inclusion degree of a class in a superclass. Consider class *Vehicle* defined as having a body and a motor and seats, represented by aggregating these parts as <*body, motor, seat*>. A tree which has a body (trunk) is much different from a vehicle, so is a theater saloon with seats. The similarity-based model assumes that the class/subclass relation is built logically at the beginning and such a class/subclass relation will probably not be built in the database. However, it is always preferable that the model accounts for arbitrary modeling. We do not include the details here, but the interested readers can find the details of the FOOD model in [16,17].

3 Conceptual Modeling of Geographic Information Systems (GIS) Applications

A GIS application usually consists of a set of maps, which include a set of geographic objects [2,3,10,12,14]. We assume that the map is modeled here as an abstract type with two basic attributes *name, scale*, i.e. the map of *East Black Sea*, with the scale *1/5000*, and a set of geographic objects. Figure 4 shows abstract type *Map* and its attributes.

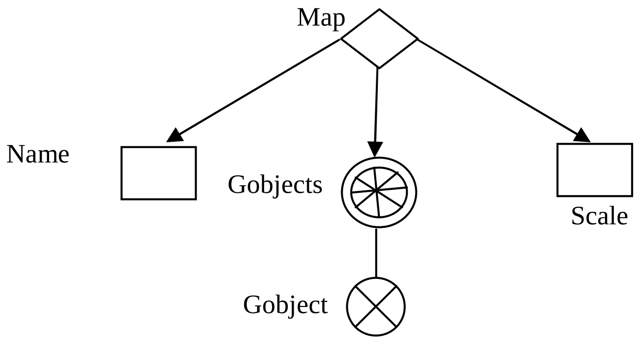

Fig. 4. The *Map* Abstract Type

Geographic objects (*Gobjects*) are a list of the *Gobject* type. Type *Gobjects* is modeled by using a *collection* constructor to group the *Gobject* objects, but being mutually exclusive, because we want to have the uniqueness constraint on the objects that it represents. *Gobject* may have three different properties. They are

GraphProp, *Description*, and *Geos*. These are aggregated to form type *Gobject* by using an *aggregation* constructor. This is intuitive because in real-life maps, for instance, the streams of a country also may have a number of attributes.

More specifically, some of graphical properties for displaying maps include *color, lnwidth,* ant the others, depending on the application. Descriptive properties may include *length, current speed, etc.,* of each stream. An example for the spatial properties that represent the structural geometry of the streams is *lines* for streams. *Gobject* may include all these properties. Figure 5 shows how to model a geographic object in the ExIFO$_2$ model by including only some of its properties.

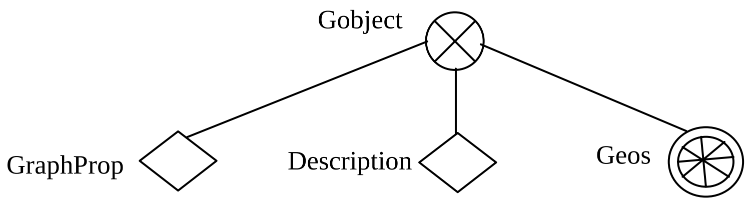

Fig. 5. *Gobject* Type in GIS Applications.

Graphical properties refer to the displaying of these objects, for instance, *color, texture, font, text, histogram* or *linewidth* of the objects. Therefore we include an abstract graphical type which consists of these attributes. The color of an object on the map can be fuzzy valued such as *dark blue, almost green,* etc. The texture may also be fuzzy valued, such as *rectangles, circular lines*, etc. *Linewidth, font, histogram* (the density of the colors in the object) and *text* are normal printable types. Figure 6 shows the modeling of *GraphProp* in the ExIFO$_2$ model.

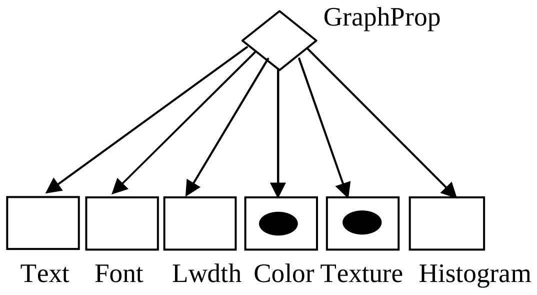

Fig. 6. Graphical properties of *Gobject.*

The *Description* type of *Gobject* is optional, because, as descriptive information, spatial information is usually sufficient. Because the descriptive information is optional for *Gobject*, we have a partial function between *Description* and *Aspatials (non-spatial)* meaning that sometimes there may not exist description. *Aspatials* is modeled as a type that is a grouping of *Aspatial*. Here we used a *grouping* constructor, because an object may have a number of descriptive entries. For instance, for cities (a city is an object of type *region*), we have at the same time, *name*(*StrAtr*), *population*(*FuzzAtr*), *lattitude*(*FuzzAtr*) and *area*(*IntAtr*). To handle this situation, the description type is modeled by a *grouping* type, which is shown in Figure 7.

Aspatial types [2] must have names, such as *population, soil, vegetation* etc. and there must exist some values as instanses of these types. In mosts cases there are three subclasses inherited from type *Aspatial*. These are *IntAtr, StrAtr* and *FuzzAttr*. An example for *IntAtr* may be *name* of a city, an example for *FuzzAttr* may be the *population* of a city. Other than these values we sometimes may also need incomplete values such as *latitude*, specified as a range 36-42. In order to handle such types, we use the *alternative* constructor which enables us to use either the crisp value for an attribute or the incomplete value, but not both. *StrAttr* is the same as *IntAttr*. For instance, the type of *landuse* is represented by *StrAttr*. *Landuse* is the *name* for descriptive information and *urban, agriculture, brushland, forest, water, barren* are values for *landuse(StrAtr)*.

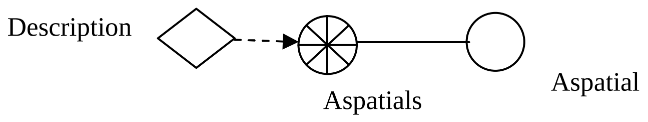

Fig. 7. The *Description* Abstract Type

A geographic object must have some spatial properties, because actual maps consist of many geometric objects. Spatial properties of *Gobject* are the geometric structure of the objects in the map, (*lines*, *regions*, *points* etc.). In the ExIFO$_2$ model we represent the spatial properties by a *collection* type, *Geos*, which handles all the geometrical properties for an object. The reason for using the *collection* constructor is that the Geo objects are mutually exclusive. The cities and the roads between these cities can be modeled by using *Geos* objects, which contains *regions, lines* and *points*. *Regions* denote the boundaries of cities, *lines* denote the roads, and *points* denote small cities. *Geo* objects are instances of the *Geos* type, which is shown in Figure 9.

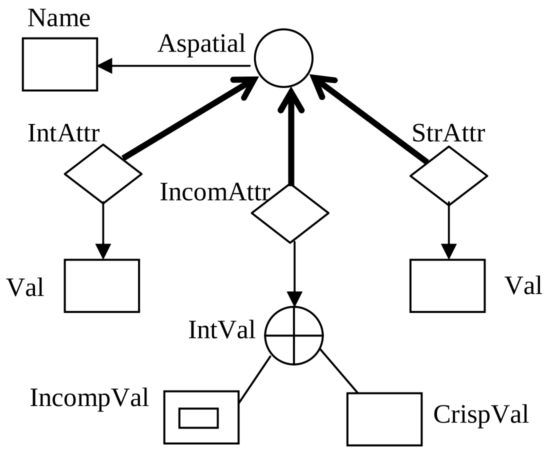

Fig. 8. The *Aspatial* Free Type

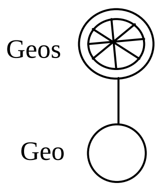

Fig. 9. The *Geos* Type with *Collection* Constructor

Geo is a generalization of *Region, Line* and *Point*. For modeling single objects, the fundamental abstractions are *point*, *line* and *region*. *Point* represents the geometric aspect of an object for which only its location in space, but not its extent, is relevant. For example a city may be modeled as a *point* in a map. A *line* is a basic abstraction for roads, rivers, cables for phone, electricity etc. A *region* is an abstraction for an object having 2d-space, e.g. a country, a lake or a national park. There are no objects other than those in *Geo*, so there are generalization links from these to the *Geo* type. Figure 10 illustrates the structure of a *Geo* object.

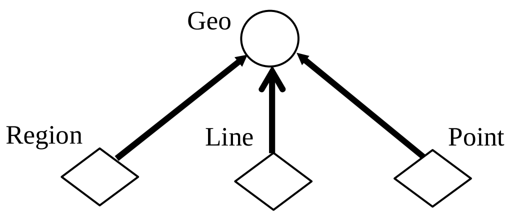

Fig. 10. The *Geo* Free Type and its Subclasses

Region is the basic type with attribute *regid*. There is a relationship between *region* and *line***,** because every region has more than one line. This is modeled by complex total function between the *region* and *line* types in the ExIFO$_2$ model. Conversely, every *line* may belong to zero or more regions. Therefore we use a partial complex function between type *line* and type *region*. These relationships are shown in Figure 11.

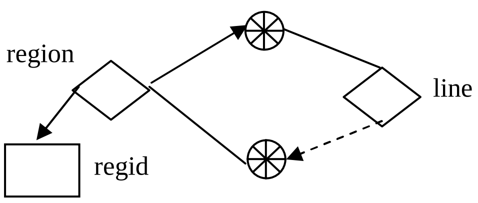

Fig. 11. The *region* and *line* Abstract Type and its Relationships

Some regions may have holes in them (lakes in a map). So we have another type *regionWithHoles* which is inherited from *region*. It has attribute *numofholes***,** which represents the number of holes in a region. This is shown in Figure 12.

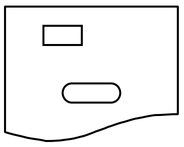

Fig. 12. An Example for *RegionWith Holes* Concept

Abstract type *line* has an attribute *lineid* and it has a construct *BegEnd*. The *BeginEnd* construct consists of two *points*: one is for the beginning of the line and the other is the end of the line. These *begin* and *end* types are of type *point*, so there is a ISA relationship between these types and type *point*, this is resulted from the specialization process (See Figure 13).

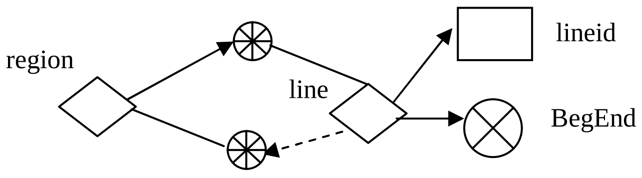

Fig. 13. The *line* Type and its Relationship with the *region* Type

Point is one of the basic constructs of type *GeoCons*. It has a *pointid* attribute and the *X* and *Y* coordinates. The *Coord* type is the aggregation of *X* and *Y,* as shown in Figure 14.

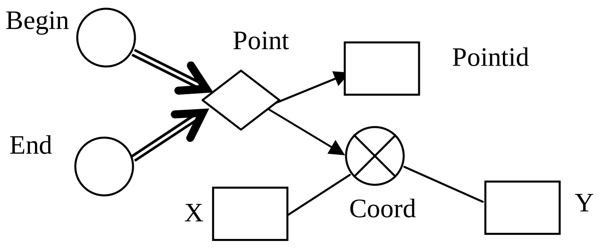

Fig. 14. The Point Type with its Related Types

Figure 15 shows the overall modeling of the GIS applications by utilizing the ExIFO$_2$ model. In the following section we outline the mapping algorithm.

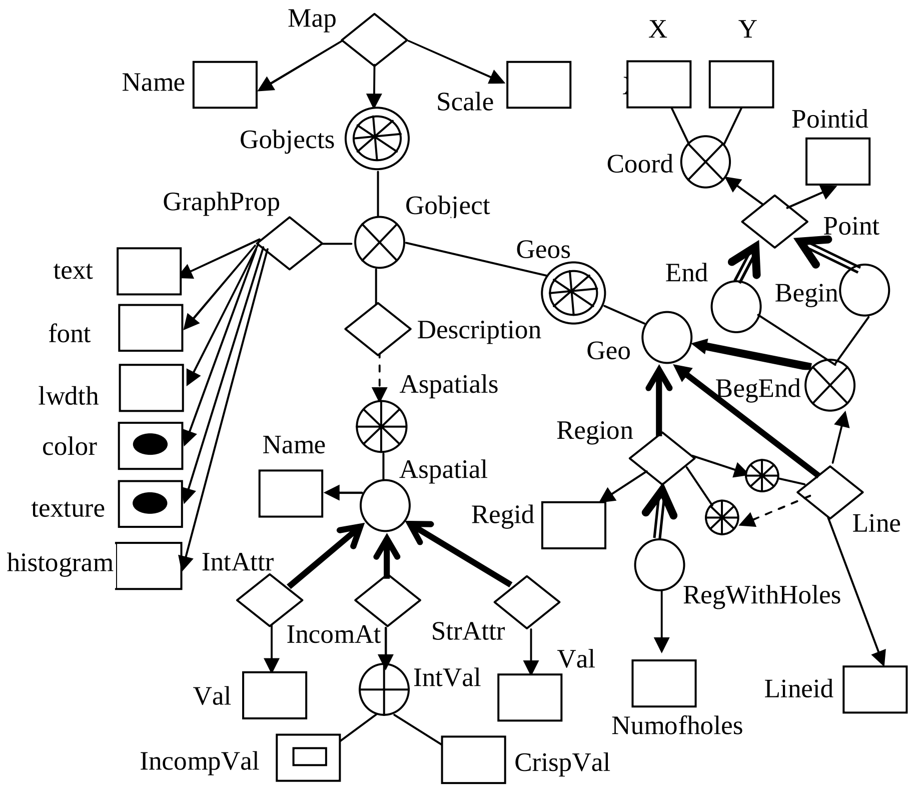

Fig. 15. Conceptual Schema of GIS Applications

4 The Mapping Algorithm

The mapping algorithm presented here transforms the ExIFO$_2$ conceptual schema into the FOOD logical database model. The algorithm is straightforward and preserves most information represented by the conceptual model.

The mapping algorithm consists of transforming the constructors of the ExIFO$_2$ conceptual model into the FOOD concepts. We first show the mapping algorithm for mapping these constructors, which include aggregation, composition, collection, alternative, and sequence, below:

```
If constructor is aggregation or  composition
  create the class name
  create the attributes of the class
  if attributes of type multivalued, incompletevalued or  nullvalued exist
     inherit from class FUZZY
     for  multivalued, incompletevalued, or nullvalued attributes (if exists)
                create the type definition for ranges and relevance
     in the class constructor
                define the ranges, relevances, semantics
if constructor is composition
                add a method to check exclusivity

if constructor is  grouping or  collection
  create the class name
  create the attribute of the class
  if attribute is of type  multivalued, incompletevalued or nullvalued
        inherit from class FUZZY
        create the type definitions for range and relevance
  in the class constructor
            define the range, relevance, semantics
if constructor is collection
  add a method to check exclusivity

if constructor is alternative
  create the class name
  create the attributes of
  create additional boolean attributes for each component
  if attributes are of type multivalued, incompletevalued or  nullvalued
     inherit from class FUZZY
     create the type definitions for ranges and relevances
     in the class constructor
        define the ranges, relevances, semantics
        assign boolean attributes to false

if constructor is sequence
  create the class name
  create the attribute of the class
  if attribute is of type multivalued, incompletevalued or nullvalued
```

```
inherit from class FUZZY
for the multivalued, incompletevalued or nullvalued attribute (if exists)
   create the type definitions for ranges and relevances
in the class constructor
   define the ranges, relevances, semantics
```

The types supported by the $ExIFO_2$ conceptual model are also transformed into the FOOD structures for implementation. These types basically consist of *atomic*, *free* and *abstract* types. Since the atomic types correspond to the properties of the objects, their mappings are trivial. But for the *free* and *abstract* types, we show how there are mapped into the FOOD model is as follows:

```
if type is free or abstract
  create the class name
  create the attributes for the components directed by the fragments
  if attributes are of type  multivalued, incompletevalued or  nullvalued
    inherit from class FUZZY
    for the multivalued, incomplete or nullvalued attribute (if exists)
      create the type definition for ranges and relevances
    in the class constructor
      define the ranges, relevances, semantics
```

Since ISA relationships are supported in the $ExIFO_2$ model, they also have to be checked and the necessary inheritance mechanism should incorporated into the FOOD model. Similarly, each fragment in the conceptual model has to be considered and mapped into the logical correspondences.

```
check for ISA relationships
   inherit from the class according to the type of ISA relationship
for the consistency of the fragments type
   add a method for checking consistency of fragments
for each of the multivalued attributes
   create the similarity matrix
```

Note that we create the similarity matrix for each multivalued attribute. If the multivalued attribute does not have fuzzy values, but only crisp values, then the identity matrix is assumed (note that the similarity relation is the generalization of the identity relation.)

Now we can map the conceptual model, shown in Figure 15, into a logical database model by applying this mapping algorithm. As we have mentioned before, the logical database model that we utilize here is the fuzzy object-oriented database (FOOD) model [16,17]. The resulting templates of the logical FOOD model after applying the mapping algorithm to the conceptual schema of the $EXIFO_2$ model for GIS applications is given in Appendix. We only give the result of the transformation without any further explanation about the resulting structures, since it is straightforward.

5 Conclusion

Nowadays, modeling needs for new applications and flaws in the currently used models have led to the definition of more powerful conceptual models. But, these models have still a number of serious limitations in implementing an increasing number of real-world applications that involve not only complex information (i.e., spatial), but also uncertainty and fuzziness. As a consequence, current research in this area is focusing on both handling fuzzy and complex information and defining new modeling and design approaches that are able to satisfy the needs for both traditional and advanced applications, such as GIS applications.

In this study, we mainly described a conceptual modeling approach for representing complex and uncertain geographic information by using an object-oriented paradigm. A conceptual schema specification for GIS applications is presented by utilizing the ExIFO$_2$ model. This model attempts to preserve the acquired strengths of semantic approaches, while integrating concepts of the object-oriented paradigm and fuzziness by including new constructors. Future work includes the design and development of a real geographic application by incorporating more geographic properties, continuous fields, and data input units and result presentation facilities into the model.

Acknowledgements

We thank to the anonymous reviewers of this paper for their helpful input.

References

1. Abiteboul, S. and R. Hull, “IFO: A Formal Semantic Database Model,” ACM Trans. on Database Systems, Vol. 12, No.4, 1987, pp. 525-565.
2. Gunther O., Riekert W.F., “The Design of GODOT: An Object-Oriented Geographic Information System,” IEEE Data Engineering Bulletin 16, No.3, 1993.
3. Guting R.H., “An Introduction to Spatial Database Systems,” Special Issue on Spatial Database Systems of the VLDB Journal, Vol. 3, No.4, Oct. 1994.
4. Hadzilacos, T. and N. Tryfona, “An Extended Entity-Relationship Model for Geographic Applications,” SIGMOD Record, Vol. 26, No.3, 1997, pp:24-29.
5. Hammer, M. and D. McLeod, "Database Description with SDM: A Semantic Database Model,", ACM Transactions on Database Systems, Vol.6, No. 3, Sept. 1981, pp.351-386.
6. Loucoupolous P, Zicari R., Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development, Wiley Professional Computing, 1992.
7. Mohan L and Kashyap R.L., “An Object-Oriented Knowledge Representation for Spatial Information,” IEEE Trans. on Software Engineering 14, No.5, 1988, pp:675-811.

8. A. Motro, "Accommodating Imprecision in Database Systems: Issues and Solutions," SIGMOD RECORD, Vol. 19, No.4, 1990, pp. 69-74.
9. Poncolet, P., M. Teissere, R. Cicchetti, and L. Lakhal, “Towards a Formal Approach for Object Database Design”, Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993, pp. 278- 289.
10. Scholl M., Voisard A., Object-Oriented Database Systems for Geographic Applications: an Experiment with O_2, in: F. Bancilhon, C. Delobel and P. Kanellakis (Eds.), *The O_2 Book*, Morgan Kaufmann, San Mateo, Calif., 1992.
11. Shekhar, S., S.Chawla, S. Ravada, A. Fetterer, X. Liu, C. Lu “ Spatial Databases – Accomplishments and Research Needs,” IEEE Trans. on Knowledge and Data Engineering, Vol. 11, No.1, Jan/Feb 1999, pp:45-55.
12. Shekhar, S., M. Coyle, B. Goyal, D. Liu, and S. Sarkar, “Data Models in Geographic Information Systems”, Communication of ACM, Vol. 40, No.4, April 1997, pp: 103-111.
13. Vila, M.A., J.C. Cubero, J.M. Medina, and O. Pons, “A Conceptual Approach for Dealing with Imprecision and Uncertainty in Object-Based Data Models,” International Journal of Intelligent Systems, Vol. 11, No.10, 1996, pp. 791-806.
14. Worboy, M.F., “Object-Oriented Approaches to Geo-Referenced Information,” Int. J. Geographical Information System, Vol 8, No.4, 1994.
15. Yang Li, Wu J, “Towards a Semantic Image Database System,” Data & Knowledge Engineering, Vol. 22, 1997, pp:207-227.
16. Yazici, George R., Aksoy D., “Extending the Similarity-Based Fuzzy Object-Oriented Data Model”, Information Sciences (Int. Journal) Vol. 108, No.1-4, 1998, pp: 241-260.
17. Yazici, A. and R. George, *Fuzzy Database Modeling*, Physica-Verlag, Heidelberg New York, 1999.
18. Yazici, A and A. Cinar, “Conceptual Modeling for the Design of Fuzzy OO Databases” *Knowledge Management in Fuzzy Databases*, Edited by O. Pons, A. Vila and J. Kacprzyk, Physica-Verlag, Heidelberg.
19. Zadeh, L.A., Similarity relations and fuzzy orderings. Information Sciences, Vol.3, No.2, 1971, pp:177-200.

Appendix

This appendix represents the resulting class templates of the logical FOOD model after applying the mapping algorithm to the conceptual schema (shown in Figure 15) of the EXIFO$_2$ model for GIS applications.

```
typedef struct {
    AnsiString          rangecolor;
    AnsiString          rangetexture;
} RangesGraphProp;

typedef struct {
    float       RLVcolor;
    float       RLVtexture;
} RLVGraphProp;

typedef struct {
    incompletevalued <int>RangeIncompval;
}RangesIntVal;

typedef struct {
    float RangeIncompval;
}RLVIntVal;

AnsiString  colordomain[] = { }
AnsiString  texturedomain[] = { }
Float colormatrix [][]= { }
Float texturematrix [] []= { }

Class Tmap {
    Public:
                Char        *name;
                Float        scale;
                Tobjects  *Gobjects;
    TMap();
}

Class TGobjects: SET {
    Public:
                TGobject      *Gobject;
    TGobjects () {
                Gobject= new TGobject;
    }
    booelan check_collection_Gobject();
}
```

```
Class TGobject {
    Public:
                TGraphProp       *Graphical;
                TDescription     *Description;
                TGeos            *Geos;
    TGobject();
}

Class TGraphProp: FUZZY {
    Public:
                Int         Font;
                Int         lnwdth;
                Char        *text;
                Hist        histrogram;
                Multivalued <ansistring> color;
                Multivalued <ansistring> texture;
                RangesGraphProp          Ranges;
                RLVGraphProp   Relevance;

TGraphProp() {
                Ranges.rangecolor =”dark blue, blue, dark green, green”;
                Ranges.rangetexture=”rectangles, circular lines”;
                Relevance.RLVcolor=0.1;
                Relevance.RLVtexture=0.1;
                Color.semantics=”OR”;
                Texture.semantics=”OR”;
    }
}

Class TDescription {
    Public:
                TAspatials       *Aspatials;
    TDescription();
}

Class TAspatials: SET {
    Public:
                TAspatial        *Aspatial;
    TAspatials() {
                Aspatial=new TAspatial;
    }
}

Class TAspatial {
    Public:
                Char *name;
                TAspatial();
}


Class TIntAttr {
    Public:
                Int         val;
    TIntAttr();
}

Class TStrAttr {
    Public:
                Char   *val;
    TStrAttr();
}

Class TIncomAttr: FUZZY {
    Public:
                Boolean is_crispval;
                Boolean is_incompval;
                Incomletevalued<int> incompval;
                RangeIntVal        Ranges;
                RLVIntVal          Relevance;

TIncomAttr()  {
                Ranges.incompval=    ;
                Relevance.incompval=  ;
                Is_crispval=false;
                Is_incompval=false;
    }
}

Class  TGeos {
    Public:
                TGeo     *Geo;
    TGeos();
}

Class TGeo { }

Class TRegion : TGeo {
    Public:
                Int         regid;
                Tline      *line;
    TRegion();
    Private:
    Boolean is_complex (Tregion, TLine);
}
```

```
Class TRegWithHoles: TRegion {
    Public:
                Int         numofholes;
    TRegionWithHoles();
}
Class TLine: Geo {
    Public:
                Int         lineid;
                TBegEnd             *BeginEnd;
                TRegion *region;
    TLine();
    Private:
                Boolean  is_partcomp(Tline, Tregion);
}

Class TBegEnd {
    Public:
                TBegin      *begin;
                TEnd        *end;
    TBegEnd();
}

Class TBegin: Point { }

Class TEnd: Point {}

Class TPoint: Geo {
    Public:
                Int       pointid;
                TCoord *Coord;
    TPoint();
}

Class TCoord {
    Public:
                Int  x;
                Int  y;
    TCoord();
}
```