Modeling and Management of Fuzzy Information in Multimedia Database Applications

Ramazan Savaş Aygün and Adnan Yazıcı

Dept. of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, NY, USA

Dept. of Computer Engineering
Middle East Technical University
Ankara, TURKEY

Abstract

In this report, we firstly present a conceptual data model for multimedia database applications based on ExIFO$_2$ model. The ExIFO$_2$ data model is chosen as the conceptual model since it handles complex objects along with their uncertain and imprecise properties. We enhanced this conceptual model in order to meet the multimedia data requirements. In addition to uncertain and imprecise information, we present a way of handling relationships among objects of multimedia database applications. Events that might be extracted from video or audio are also considered in this study. Secondly, the conceptual model is mapped to a logical model, which the fuzzy object-oriented data (FOOD) model is chosen, for storing and manipulating the multimedia objects. This mapping is done in a way that it preserves most of the information represented at the conceptual level. Finally, in this study videos of football (soccer) games is selected as the multimedia database application to show how we handle crisp and fuzzy querying and retrieval of fuzzy and crisp data from the database. A program has been developed to draw ExIFO$_2$ schemas and to map the schema to FOOD code automatically.

1 Introduction

In recent years, management of multimedia information has been the focus of many applications such as video on-demand services, digital libraries, museums, on-line shopping and document management. These applications require development and design of new databases where the multimedia data is the resource. Multimedia databases are expected to fulfill the requirements of users for efficiently and flexibly developing and managing multimedia applications. The development of multimedia database systems is composed of three levels as in traditional databases. These three levels are conceptual level, logical level and physical level.

The design of multimedia databases differs from that of traditional databases due to the characteristics of multimedia objects, such as image, graphics, audio, and video. As pointed out in [9], one of the challenging issues that multimedia database researchers have to face is the development of conceptual models for multimedia information especially for video, audio, and image. The conceptual model should be rich in its semantic capabilities to represent complex media objects. Following the conceptual data model, transformation from the model to a logical database schema is required for storing and manipulating these objects.

Until now, many conceptual approaches are proposed to model complex data [5, 12, 17, 22] at conceptual level. Since the conceptual models were developed for domain-specific applications, further steps should be taken to enhance conceptual models to represent the complexity and requirements of multimedia objects. There have been a number of attempts to develop conceptual models for representation of multimedia objects. In [9], these models were classified into several categories; graphical models, Petri-Net based models, and object-oriented models.

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 multimedia objects. For instance, attributes of an image such as color, description of an object and spatial relations between objects may be interpreted in various ways. As a consequence of these, queries involving imprecise and uncertain information may be unavoidable. Consider the following queries:

- **Q1: "Find all hand images of** *pre-teen* **males that are** *abnormal*"
- **Q2: "Retrieve images that have colors** *similar* **to a sunset photograph".**
- **Q3: "Retrieve all facial images with a** *short round* **chin and** *thin* **hair"**
- **Q4: "Retrieve an object** *partially-surrounded-by* **a** *little* **of object A"**
- **Q5: "Retrieve videos of all goals scored by Shearer** *very near* **to the goalpost"**

Q1 is an example query in a medical image database. In Q1, *pre-teen* and *abnormal* are fuzzy templates. In Q2, *similar-to* is a fuzzy operator, which has to employ similarity matching. Q3 is a query for facial image database in which *short round* and *thin* are fuzzy concepts. Q4 is an example for Geographic Information Systems that contains fuzzy concepts such as *partially-surrounded-by* and fuzzy quantifier *little.* Q5 is an example of query that might be needed in a video. Here *near* is a fuzzy concept and *very* is a fuzzy modifier. If these kinds of uncertain, imprecise, and fuzzy information are forced to be precise, then there might be some loss of information. Uncertainty in databases has been studied by many researchers such as [15, 23].

However, there has not been much research on the development of conceptual models which deal with uncertainty for multimedia databases. In this study, the conceptual model that we present for multimedia database applications is based on ExIFO$_2$ data model. ExIFO$_2$ [22] is a fuzzy object-oriented conceptual data model that includes representation of fuzzy information and handles object-oriented concepts. This model attempts to preserve the acquired strengths of semantic approaches, while integrating concepts of the object-oriented paradigm and fuzziness. ExIFO$_2$ model supports uncertainty at attribute level as well as the class level. It includes three types of uncertainty at the attribute level: fuzzy, incomplete and null valued attributes. After constructing a data model for multimedia database applications, we also show how the ExIFO$_2$ schema is mapped to the object-oriented data model, FOOD [24, 23].

One of the main focuses of this study is conceptual modeling of media where the multimedia objects are presented. In this study, conceptual modeling of audio, video and image is performed. Another focus is the modeling of relationships among objects in multimedia data. These relationships can be either spatial in an image or real-life relationships (roles) which might not be extracted automatically from the video. Modeling of mapping from real objects to multimedia objects is also performed. We also designed events that might take place in a video/audio since most of the applications require querying according to the events.

As a multimedia application, we have chosen videos of football (soccer) games. The developed conceptual model is appropriately adapted to this application. Then the conceptual model is mapped to FOOD [24]. A user interface is prepared for querying the database. A program which enables drawing of ExIFO$_2$ schemas and enables querying on videos of football games is built. The user can draw and modify ExIFO$_2$ and map the schema to FOOD code if he/she desires. The user can perform both crisp and fuzzy queries.

This report is organized as follows: The following section discusses uncertainty and object-oriented databases. In section 3, multimedia database applications and issues are explained briefly. In Section 4, conceptual data modeling of multimedia database applications along with ExIFO$_2$ is explained and then its application for a multimedia database application, videos of football, is presented in Section 5. Section 6 describes how the mapping is performed from ExIFO$_2$ model to FOOD model. The querying process is presented in Section 7. The last section concludes our work.

2 Uncertainty and Object-Oriented Databases

The management of uncertainty in database environment is necessary in application areas such as multimedia databases, engineering applications, decision problems, control systems, and knowledge based systems. Most of the studies on fuzzy databases are developed by either using or extending the ordinary relational models. There are only few studies to handle uncertainty and fuzziness under object-oriented databases [24, 15]. Fuzzy and uncertain object-oriented databases support rich semantic expressiveness.

According to fuzzy set theory, each element has a membership to a fuzzy set that it belongs to. Membership values lay in the range [0,1]. If the membership value is high, it is more likely to be member of the set and if it is low, it is less likely to be member of the set. The functions which determine the membership values of elements to sets are called membership functions. The notation for expressing membership of element x to set A is denoted as $\mu_A(x)$. The relationships between fuzzy sets are investigated in [25].

The similarity-based fuzzy object-oriented data model (FOOD) [24] facilitates the enhanced representation of different types of imprecision. Similarity relation has the basic characteristics of the similarity relations that are contained in [25]. In FOOD, similarity relation allows impreciseness in data to be presented and imprecision in data results in uncertainty in classifications. The similarity relation is both a generalization of equivalence and an extension of the degree of membership concept for set elements. Here the domain elements are considered as having a varying degree of similarity, replacing the idea of exact equality/inequality. Each database domain, D, has an associated similarity relationship (represented as a matrix) that assigns a value between 0 (zero) and 1 (one) (including 0 and 1) to each pair of domain elements. That is, the membership grades are not associated with individual data items (as it is for fuzzy membership values) but with entire classes. In this model, a similarity matrix is kept for each fuzzy attribute to declare the similarity between the values under the corresponding domain. For example, if the attribute is *color*, then the contents of similarity matrix are the similarity of *colors* and similarity among these values are in the range [0,1].

2.1 Attribute Level Uncertainty

In the FOOD model, three types of uncertainty are distinguished at the attribute level, *fuzzy*, *incomplete*, and *null.*

2.1.1 Fuzzy Attributes

Fuzziness exists when true data is available in descriptive terms rather than as a precise data. These imprecise values can be related to each other with AND, OR, and XOR semantics. By using AND semantics, the book of a color can be represented with <red, blue> , i.e., "the color of the book consists of two colors, red and blue". In fuzzy sense we may interpret this as, one of the colors is red (or a very similar to red, i.e., redish) and the other is more or less blue. By using OR semantics, the face expression of a person can be represented with {worried, sad}, i.e., "the expression of his face may be worried or sad." In fuzzy sense the expression of the face is vague, either worried or sad or another expression similar to these. By using XOR semantics, a kick in a football game can be represented with [free kick, indirect kick], i.e., "the kick is either free kick or indirect kick, but not both". There is an uncertainty involved here since one cannot always distinguish whether the kick is a free kick or indirect kick, but only one of them must be chosen for each specific kick.

The database should allow queries including both fuzzy and precise values. Therefore, in the database, if the data is acquired in precise form then it is kept exactly. Fuzzifying the precise data and storing the fuzzy value would cause loss of information. In contrast, ignoring fuzzy value in absence of precise data would also cause loss of information. Comparison of fuzzy and precise values is handled by membership functions. Each value of a fuzzy attribute has a corresponding membership function. For example, the domain of height is {short, normal, tall}. The membership degrees of a precise height value (e.g., 1.75 cm) are calculated for each of short, normal and tall.

Queries might also include threshold values to restrict the objects to be selected. If the threshold is high, only objects which are the most similar are retrieved. Decreasing threshold causes additional retrieval of objects.

The *inclusion formula* yields the similarity between two imprecise values. The inclusion value differs as the semantics between the values change. The computation of inclusion formulas for AND, OR, and XOR semantics has been explained in [24].

2.1.2 Incomplete and Null-valued attributes

Another type of uncertainty is *incompleteness* which occurs when true data may belong to a specific set of values. For example, in a movie, the main character has the same attributes as a person. He has a *name, age,* etc. But we might not know the exact age of this person. Instead, we can predict his age as a range such as between 25 and 35. The last type of uncertainty at the attribute level is *null*. If an attribute's type is *null*, this can imply one of the following: the attribute's value does not exist or it exists but its value is not known or it is not even known whether the attribute's value exists. Consider an example of an event that takes place in video. We might have no information if the event has a reason or not, or we might know that the event has no specific reason or the event has a reason but we might have no information about what it is. The computation of inclusion formulas for incomplete and null-valued attributes is explained in [22].

2.2 Class/Object Level Uncertainty

The second level of uncertainty exists on the object and class levels. This uncertainty represents the membership degree of an object to a class. For example, in an image database which contains landscape images, all images can be classified into three classes: forest landscapes, sea landscapes and urban landscapes. The image of a village which is by the sea and next to a forest can be an object of all the landscape image classes with different membership degrees. If we consider only the color histogram of the image, we can assign a different membership degree ([0,1]) to each of the classes according to dominating colors. The computation of inclusion formulas for class/object level has been given in [24].

2.3 Class/Subclass Level Uncertainty

The degree of membership of a class to its superclass can vary. For example, class video-insects can be subclass of video-winged-animals with a membership degree in [0,1]. This type of membership introduces the notion of *fuzzy classes*. There may exist uncertainty also between a class and its subclasses. It is not always possible to construct class hierarchy precisely because of the conceptual distance between the class and its subclass. Fuzziness can be used at the class/subclass level and the system can be queried according to this uncertainty. The system can answer queries about membership of classes to its superclasses such as "To what extent a class belongs to its superclass?". The membership degree of a class to its superclass can be determined by using the range definitions of classes as in object/class level. The detailed explanation of the computation of the inclusion formula exists in [22].

3 Multimedia Database Applications

Multimedia Objects (MOBs) are usually complex objects composed of other objects that can also be complex. The object-oriented model is one of the best models to express the object hierarchies. MOBs are audio-visual in nature [13]. This results in imprecise and incomplete description of MOBs and subjective interpretation of the users. This is the case of MOBs where uncertainty and fuzziness needs to be embedded. It is clear that the user has no idea about how MOBs are stored in the database. The language that the user uses to retrieve multimedia objects or data might consist of uncertain, incomplete, and fuzzy expressions. MOBs are multidimensional and hierarchically structured, and can have some relations among them. Temporal relation is related to the time that objects become active. Some of the basic temporal relations can be stated as *during, before, overlapping, meet,* etc. The spatial relation is related to the distribution of the objects in space. Some of the basic spatial relations are *left of, behind, inside,* etc. These two types of relations are expected to exist in multimedia databases to handle content-based retrieval.

Most research on media focused on image, video, and audio. Video is a sequence of images. Each image in a video is called a video frame. Video has both temporal and spatial behavior. Video has temporal behavior in the sense that image sequences of the video should be displayed in order and in some dedicated time. Audio has temporal behavior in the sense that it should be played in order and in some dedicated time. Audio/Video is combination of audio and video where it might be necessary to play video and audio in harmony. A video clip is a subset of the sequence of images that form the video such that it has meaning and events that can be defined. The number of video frames does not have to be constant and events can be extracted from variable number of sequential images. An audio clip is some partition of the audio and it has the same characteristics as described above for a video clip. We assume that each video clip has a corresponding audio clip. An audio clip can also be empty, that is, it does not exist or can not be played.

In [7], a graphical data model for video data which supports spatio-temporal semantics is proposed. Video is composed of clips, sequences of frames. Each clip is partitioned into segments. Image data model is described as a scheme for representing objects in images, their visual and geometric characteristics such as color, texture, spatial and topological relationships between objects and semantics associations such as aggregation, generalization/specialization among objects [11]. Yang and Wu [20] developed a semantic image database on top of the IFO [1] conceptual data model to describe inner structure and contents of images. OVID [14] is a prototype video-object database system. OVID offers schemaless description of database, interval inclusion inheritance, and composition of video-objects based on IS-A hierarchy. In [2], a way of organizing and structuring video data to facilitate queries is described. There are two concepts which needs to be retrieved from the queries: entities and video frames. Entities can be video objects which are present in video frames or activities which are subjects in the sequence of video frames. Events can be considered as instantiation of activities with objects that take part in the activity. Events can be distinguished from each other with the objects involved in them. Roles are descriptions of activities and team is the set of all objects/descriptions that jointly describe an event. Fuzzy queries in multimedia database systems have been studied in [8]. Only one uncertainty of attribute level (*fuzzy*) has been allowed and the comparison of values at different levels of uncertainty has not been considered.

Data model should enable relationships such as PART-OF and IS-A relationships. PART-OF relation simplifies the management of complex objects and IS-A relation handles the class hierarchy. In addition to above, SIMILAR-TO relationship which enables imprecise matching objects is an important relationship that should be incorporated in a multimedia database management systems. A spatio-temporal semantic model for multimedia database systems using augmented transition networks (ATN) has been proposed in [6]. The temporal layout and spatial layout of multimedia objects are represented with multimedia strings. In [7], the user can define his or her own view of the database in an object-oriented way. There are three types of objects the user should deal with: conceptual spatial object (e.g., sitting), conceptual temporal object (e.g., walking, slam-dunk) and physical objects (e.g., persons, trees, houses). In [19], the scenario, events in the context of a multimedia application, consists of the actions that take part in multimedia application and the response of the applications to the events. The conceptual modeling of multimedia database applications using ExIFO$_2$ model has been studied in [4]. The different levels of uncertainty has been modeled for multimedia objects.

4 Conceptual Data Modeling of Multimedia Database Applications

In this section, a conceptual data model for multimedia database applications is built using ExIFO$_2$ data model [22]. The EXIFO$_2$ model is a formally defined conceptual database model that comprises a rich set of high-level primitives for database design. More formally, an EXIFO$_2$ scheme is a directed graph with various types of vertices and directed edges, representing atomic objects, constructed objects, fragments and ISA-relationships. Since the formal definitions of EXIFO$_2$ components are given in [24, 21], we only briefly summarize the related important concepts here.

Atomic objects are the basis of any IFO schema and involve three types: *printable*, abstract and free. *Printable* objects correspond to objects of predefined types that serve as the basis for input and output. Abstract objects correspond typically to objects in the real world that have no underlying structure. The third type of atomic object is called free, and corresponds to entities obtained via the "ISA" relationship. Constructed objects are composed of underlying object representations by a finite set of grouping or aggregation constructors.

4.1 Using ExIFO$_2$ for Multimedia Database Applications

In this study, we utilize ExIFO$_2$ data model for conceptually modeling multimedia database applications. Rather than considering every type of media, we concentrate on video, image, and audio. Video has both temporal and spatial behaviors, image has spatial behavior and audio has only temporal behavior. Our model is extensible and have the power of representing other structures of multimedia applications. We include uncertain and incomplete information as well as fuzzy classes. In the following subsections, how to handle types, constructors and fragments of ExIFO$_2$ data model and some additional constructors to form complex objects from multimedia data are discussed along with examples.

4.1.1 Types

There are three basic atomic types in ExIFO$_2$ data model: *printable, free*, and *abstract.* The file format, such as *bitmap, gif*, of an image can be considered as *printable* type. The image can be an abstract type. The video frame is an example of a free type in the model. The representations of these types are depicted in Figure 1 (a).

To handle uncertainty at the attribute level, there are three types: *incomplete-valued, null-*

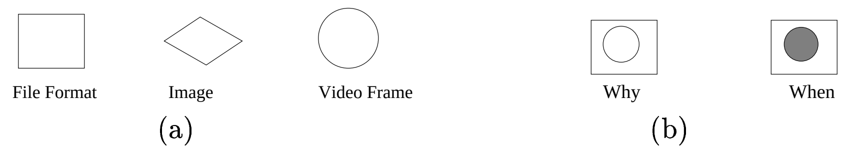


Figure 1: (a) Basic Atomic Types (b) Uncertain Atomic Types

valued, and *fuzzy-valued*. The reason(why) for an event can be *null-valued*, whereas the time(when) of the event can be *fuzzy-valued* (Figure 1 (b)).

4.1.2 Constructors

Two of the constructors that exist in ExIFO$_2$ data model are *aggregation* and *grouping*. These constructors are also used in modeling multimedia data. For instance, *aggregation* constructor is utilized in combining image attributes such as image *title, format, width* and *height* (e.g., "Ferrari", bitmap, 256, 256) as in Figure 2 (a). These attributes can be extended according to the requirements of the multimedia application. A set of images that constitute all pictures of a certain automobile, e.g., Ferrari, can be handled by *grouping* constructor.

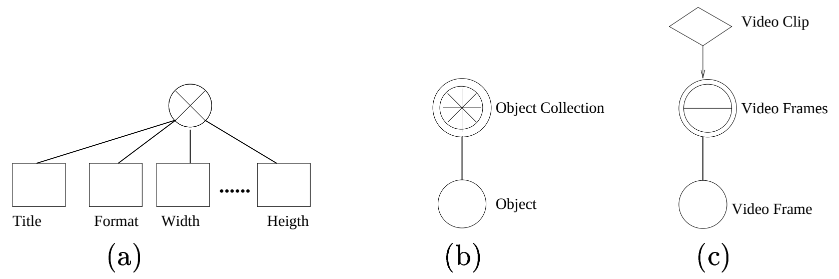


Figure 2: (a) Aggregation Constructor (b) Collection Constructor (c) Sequence Contructor

Composition and *collection* constructors require exclusivity in *aggregation* and *grouping* respectively. The *composition* constructor can be used when components of an object (e.g. image id number) can be uniquely identified. On the other hand, the *collection* constructor can be utilized to group the objects in an image (e.g., in a picture of a computer on a table, computer and table are two objects grouped using collection constructor) (Figure 2 (b)).

In addition to these constructors, a new constructor is used to improve the modeling power for multimedia applications. This is the *sequence* [18] constructor that is modified version of *aggregation* constructor and includes the chronological order of the constituents. For instance, a *video clip* is segmented into *video frames*. There exists a temporal relation between each *video frame*. A *video clip* is composed of an ordered sequence of *video frames* with respect to time (Figure 2 (c)).

4.1.3 Fragments

The types in ExIFO$_2$ data model can be linked by using functions called fragments. The goal of the fragment is to describe the properties of the principal types. The functions can be *total function, partial function, complex-total function* and *complex-partial function.* In our data model, fragments handle relations among the objects in an image. An object might have more than one relation with more than one object (e.g. an object may have two neighbors, *left-of* and *right-of*). Thus we used *total, complex-total* and *complex-partial* functions. Each object in an image must have some properties (*total function*) (e.g. color, texture). The object may have zero or more relationships with other objects (*complex partial function*). Lastly, a relation may consist of one or more objects (*complex-total function*) (Figure 5 (b)).

4.1.4 ISA Relationships

The last structural component of the ExIFO$_2$ model is the representation of ISA relationships interpreted in two ways as *specialization* and *generalization.* We use these relationships in the same manner as it is used in [22]. We use a *specialization* link to map objects in an image to the real objects (e.g. Object has the same attributes with the ObjectType1) (Figure 5 (b)). In this figure, *object* is a free type and *object types* can be free or abstract. This is a restriction set by *specialization* link property. According to the application, the types of *object types* are decided.

4.2 Modeling Multimedia Database Applications

In our modeling approach, a video is composed of name, description, pointer to raw data and sequence of clips as depicted in Figure 3 (a). Clips are composition of audio and video clips which are shown in Figure 3 (b).

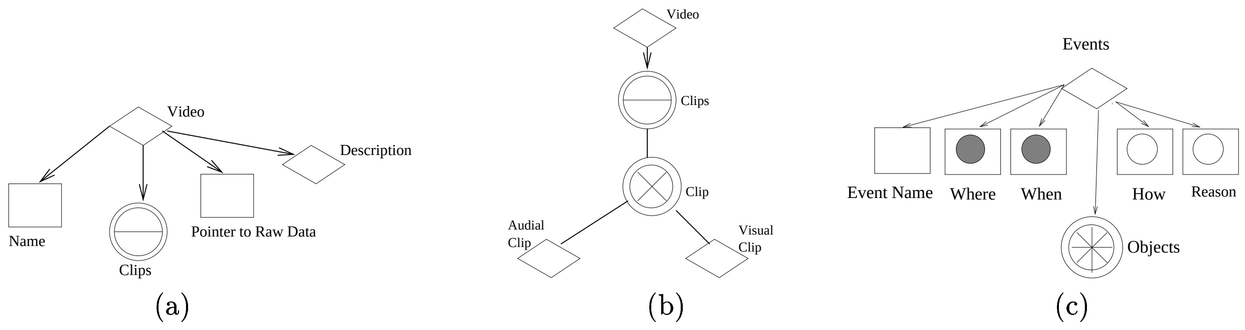


Figure 3: (a) Video Components (b) Clips of a video (c) Event

Audio and video clips have starting and ending frames which can be used for synchronization and temporal relationships. Events can be defined both for audio and video. In this work, we did not list all of the attributes of video and audio, but these can be found in [10, 16]. An event consists of *objects* that take part in an event, the *reason(why), what, how, where* and *when* it happened as depicted in Figure 3 (c). "John runs slowly in the forest to be healthy in the morning" is a possible event in a video clip. *Run* corresponds to the action (what), *in the forest* denotes where the run occurs, *John* represents the object, *to be healthy* denotes the reason for his run, *slowly* corresponds to how fast he runs and *in the morning* represents when he runs. This event structure is similar to the one in [2].

Audio clip is a sequence of audio frames. Each audio frame is a one-dimensional signal that has *frequency* and *amplitude* as physical properties and *title* and *keywords* as description of the signal. The schema of audio clip is depicted in Figure 4 (a).

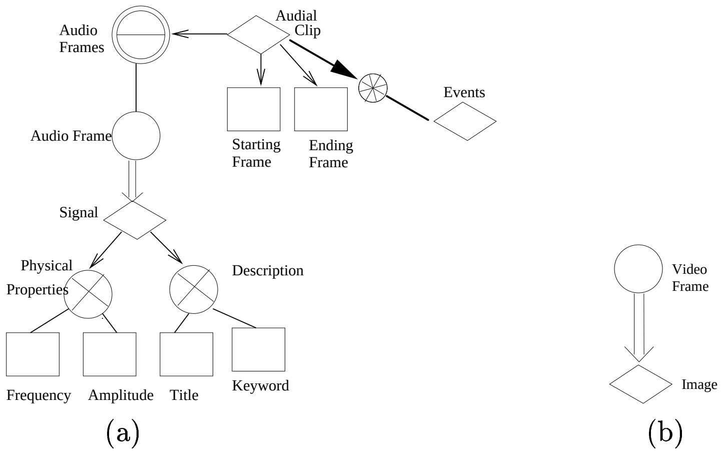


Figure 4: (a) Audio Clip (b) Video Frame

Each video clip is a sequence of video frames as shown in Figure 2 (c). The study done by Y.F.Day et al. [7] also includes a model of video clips, which is somewhat similar to our approach. In their work, each video clip is composed of segments. Each segment is distinguished from other segments in the objects that they contain. The model that they propose represents a video clip in the form of a directed graph. However, in our approach, clips are distinguished when a new event starts. In this model, each clip has at least one event. Temporal relations among objects can be retrieved according to starting and ending times of their clips. *Video frame* is an image and this is depicted in Figure 4 (b).

Image has a collection of objects and physical properties about the attributes of the image and features extracted from the image. Attributes are an *aggregation* of *title, format, width* and *height* of the image. Features of an image are *histogram, raw data, texture*, and also *fuzzy texture* and *fuzzy color* descriptions. The components of *image* are shown in Figure 5 (a).

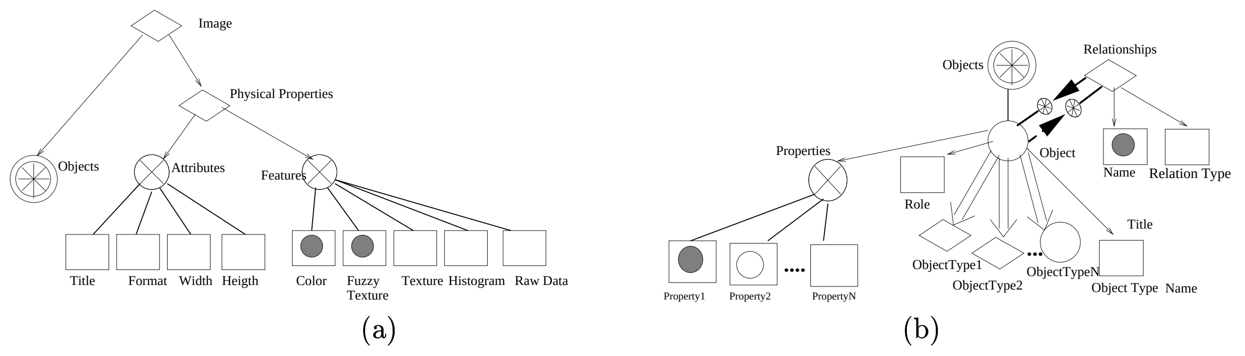


Figure 5: (a) Image (b) Object

Objects can have minimum bounding rectangles for determining its location and spatial relation with other objects in the image. Objects in a video are related to each other with a relation. A relation keeps the object and the relation type with the corresponding objects. Relation type can be *spatial* relation (e.g., *object A is partially-surrounded by object B*) or any other relation that might exist such as *family* (e.g., *A is father of B*). Objects also have roles which might differ according to the context. For example, in a video a person can be *employee* at his work during the day and a *husband* after the work. Objects have dynamic properties that can change in various images. These properties can be *color, face expression*, and *state* of the object. The types of these properties can be *fuzzy-valued, null-valued, incomplete-valued* or *printable* depending on the application. An object in the image is, in fact, a specialized version of a real time object such as *person, animal, staff*. The components of an object are depicted in Figure 5 (b).

The result of conceptual modeling of multimedia database applications is represented as a generic conceptual schema using the ExIFO$_2$ model and is shown in Figure 6. This conceptual model does not handle all of the details related to all multimedia database applications. It only gives the basics of conceptually modeling a multimedia database application. The model might need to be extended for an application or some parts from the model may be removed for other applications. Here we consider mapping from real objects to objects in multimedia application, handling relationships among objects, and handling uncertainty as the crucial parts for conceptual modeling. We use the sequence operator since there exists chronological order between the video frames of a video and audio frames of audio as well as clips of audio/video.

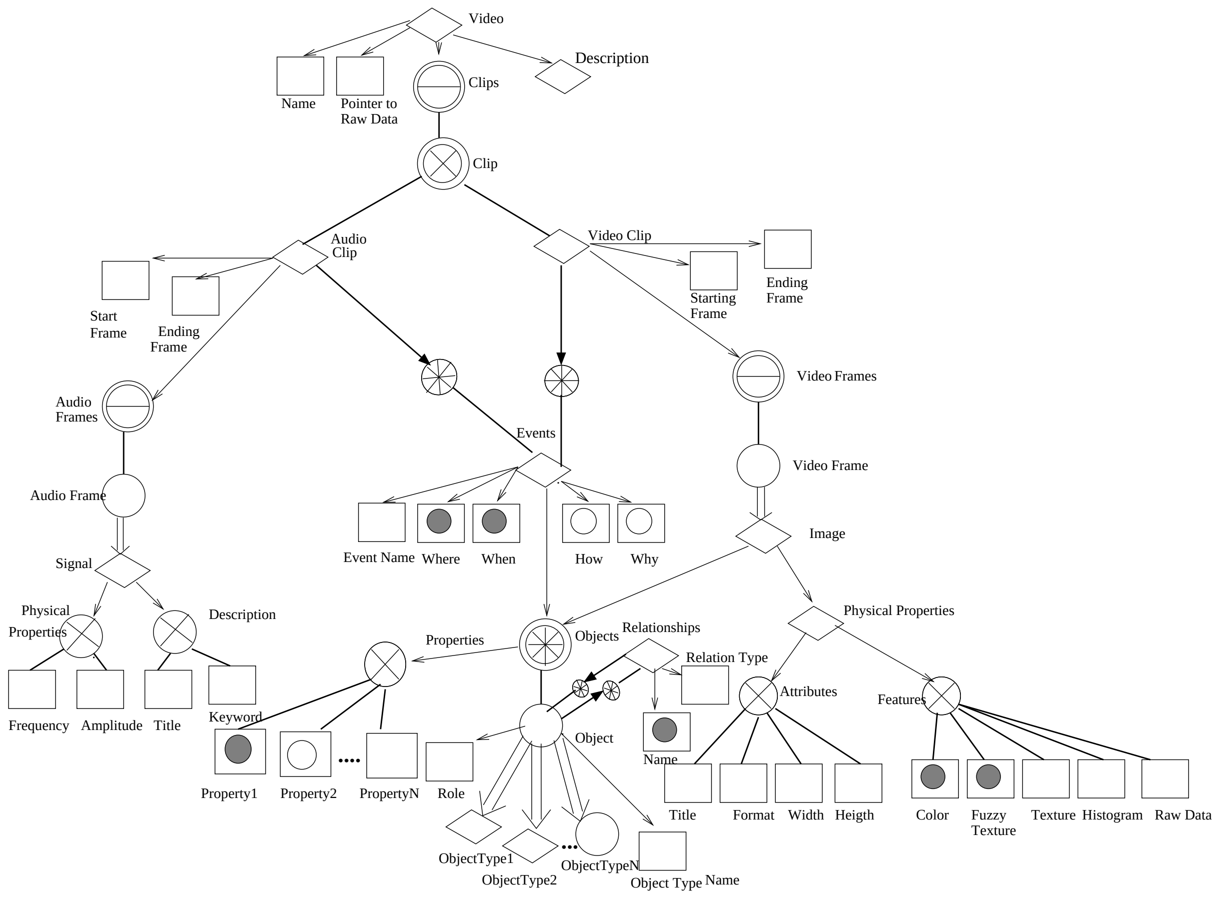


Figure 6: The Schema for Multimedia Database Applications

5 A Multimedia Database Application:Videos of Football Games

There have been various multimedia database applications that our conceptual model can be exploited. Videos of films, news, or sport games are examples of these kind of applications. As an example, in this study we have chosen a sport activity, football.

In this work, the domain of videos is a set of football games. In this section, the details of a football game and the properties that should be extracted for a football game are stated along with the conceptual model.

Each football game video has a name, a description, a pointer to raw data and a sequence of clips. The schema of this part is shown in Figure 3 (a). The name is chosen as the pair of team names since these are the basic keywords to have opinion about what the video is. For example, if the game is played between England and Germany, the name of the video can be declared as *England-Germany*.

The pointer to raw data is composed of the file name and the file location. This is needed when the results of the queries are presented to the user. The description is usually application specific. In our application, the description consists of the first team name, the second team name, the score of the first team , the score of the second team, the number of audience, the stadium where the game is played, the date of the game, the type of the game, the weather condition when the game is played. Since the audience number might not be always known exactly, this attribute is denoted with the *incomplete* constructor to state that the audience count lays in a range. The weather condition has fuzzy values such as sunny, hot, rainy, cold, snowy etc. Since these values have some degree of similarity, this attribute is denoted with the *FuzzySet* constructor. The type of the game determines whether a game is a friendly, world cup, or cup final etc. The stadium and the date of the game corresponds to the stadium where the game is played and the date when it is played, respectively. The stadium, date, team names and team scores are represented with the *printable* constructor. The description is depicted in Figure 7 (a).

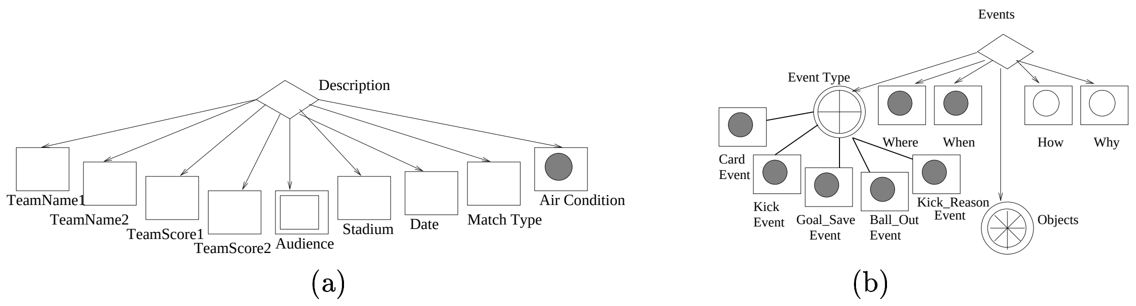


Figure 7: (a) Video Description (b) Football Event

A video is a sequence of clips, i.e., a video is partitioned into smaller parts and the video is a sequence of these constituents. This is represented with the *sequence* operator. A clip has two components: an audio clip and a video clip. A clip is a composition of these components since each audio clip or video clip exists only once along the video. This part is shown in Figure 3 (b). An audio clip is a sequence of audio frames and consists of events. In addition, it has starting and ending frames. Since audio clip has at least one event, this is depicted using *complex-total* function. Events are explained when a video clip is explained. An audio clip is a one-dimensional signal and this is depicted with the *specialization* link. A signal is composed of its physical properties and description. The physical properties are *aggregation* of frequency and amplitude. The description of signal is *aggregation* of title of the signal and keywords. The audio clip of a video with its subparts are shown in Figure 4 (a).

A video clip has also starting and ending frames. It is a sequence of video frames and a video clip contains at least one event as in audio clip. Video clip is depicted in Figure 3 (b). Video frame is an image and this is represented by using the *specialization* link. This *specialization* is shown in Figure 5 (b).

An event has 5 components: the *type* of event, the time *when* the event happened, the place *where* the event happened, the *reason(why)* for the event and *how* the event happened. There are five types of events in our application: *card event, kick event, ball_out event, goal_save* and *kick_reason* event. An event can be only one of these, so event types are depicted with the *alternative* constructor. Each of these event types are represented with the *FuzzySet* type. Possible card events are *yellow card* and *red card.* The card event has OR semantics. This means that a card event can be *yellow card*, *red card*, or *yellow card* and *red card.* In a football game, a player can be booked with a *yellow card*, or sent out of the game with a *red card* directly or he can be sent out of the game *red card* following a *yellow card.* The kick event can be a *free kick, indirect kick,* or *penalty.* In kick event, only one of these events are true and has XOR semantics. Because, a kick cannot be a free kick and indirect kick at the same time. The ball_out event can be *out* or *touch* . This event has also XOR semantics. The goal_save event has also XOR semantics since a player either scores or the goalkeeper saves it. The reason for a kick in our application can be a *foul.* This type of event has also XOR semantics. All these events have to be fuzzy in our application although at first sight they seem to be crisp. For example, when the referee can whistle a kick, it is not always possible to distinguish an *indirect kick* and *free kick.* This data should be kept as either indirect kick or free kick. Otherwise, there is a loss of information if one of them is forced. Also, the user might not be sure of the kick that he wants to watch. The *reason* and *how* the event happened are represented with the *null* constructor. *How* attribute can be *holding, pushing* etc. in case of foul event. Foul event is of type kick_reason event. The reason for a *yellow card* can be *objection to referee.* The time and the place of the event are represented with the *FuzzySet* type. Closeness of the player to goalpost can be a fuzzy term such as *near*, *far* etc. The time of a game with respect to the beginning or end of a game can also be a fuzzy term. The event schema for football is shown in Figure 7 (b).

An image has a collection of objects and has properties about the physical image. Physical image is composed of attributes and features. Attributes of an image are *title, format, width,* and *height.* Features are *texture, fuzzy texture, histogram, raw data,* and *color.* Color and fuzzy texture are fuzzy attributes. The components of image are depicted in Figure 5 (a).

Each object in the image has a type, role and relations. Type determines the kind of the object such as ball and player. The role determines object's role in an event. Each object might have relationships with other objects in the image. Since having a relationship is not a constraint, this is depicted with the *complex-partial* function. Each relation has a relation type, relation name and a related object. If there is a relation with another object, then the related object must exist. Therefore, this is denoted with the *complex-partial function.* The relation type determines the type of the relation. This can be *spatial, action, family*, etc. For example, *Player A is in left-of Player B* is a relation of type spatial. *Player A passes the ball to Player B* is a relation of type action. The components of the objects are shown in Figure5 (b).

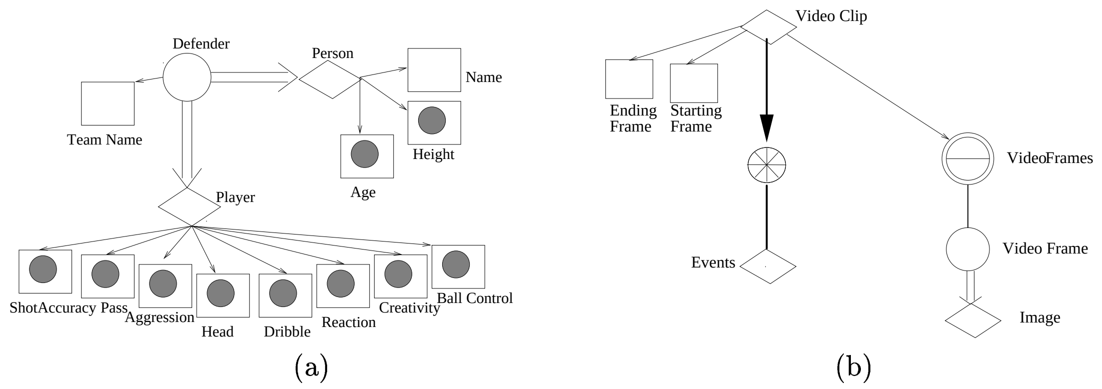


Figure 8: (a) Defender Player (b) Video Clip

Objects are *specializations* of other types. In this application, possible objects are *ball, player*, and *goalpost.* A player can be a defender player, a middle-field player, a forward player or a goalkeeper. Each of these have some player properties: *aggressiveness, heading, reaction, agility, ball control, creativity, pass, dribble, shot power, shot accuracy, team name.* The values of these properties except team name are denoted with the *FuzzySet* type since there is no metric to calculate these properties. The team name is represented with the *printable* type. Each of the defender, middle-field, goal-keeper, and forward types have domains in which some of these properties are essential. The schema for a defender player is shown in Figure 8 (a). Although a player can be e.g., a middle-field player with a some degree, he can also be a forward player with another membership degree. In this case, class/object level fuzziness occurs. Video clips along with its subparts, events, video frames, image, objects are shown in Figure 8 (b). The whole schema for the video of football games are shown in Figure 9. The details are ignored in this figure not to complicate it more.

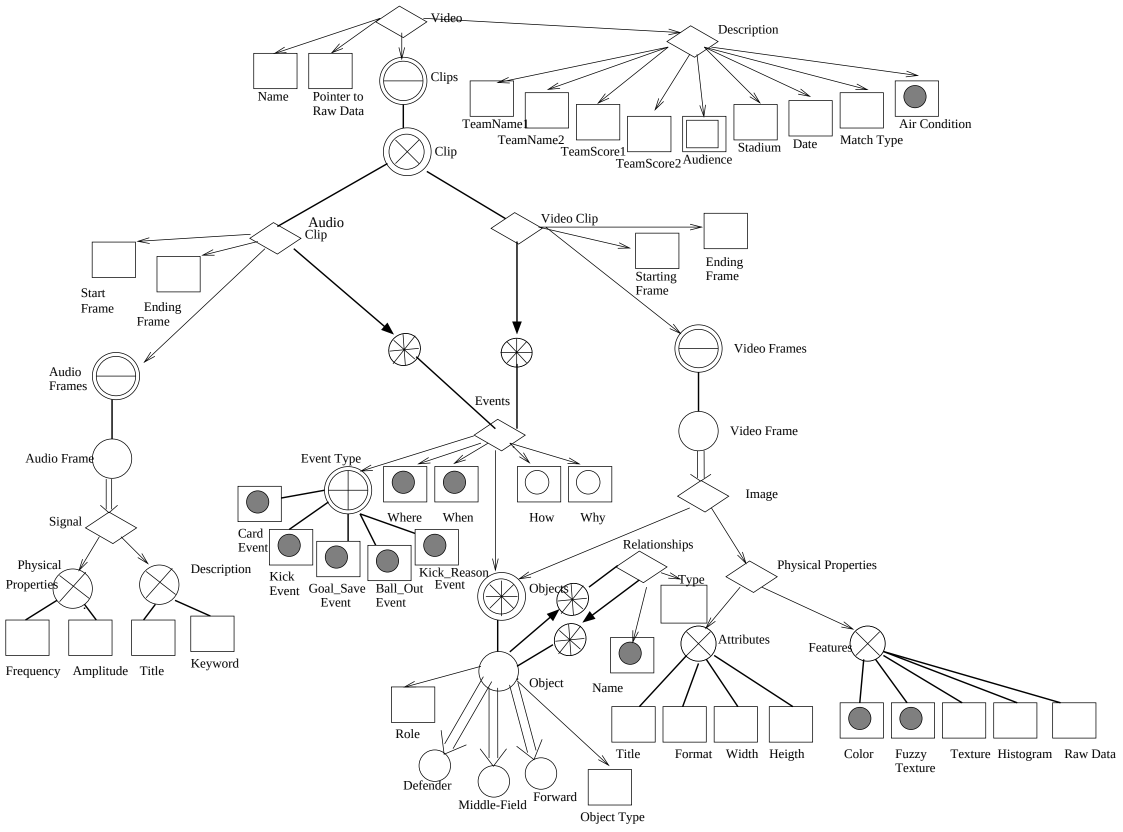


Figure 9: Schema of Videos of Football Games

6 The Mapping of the Conceptual Model into Logical Object-Oriented Model (FOOD)

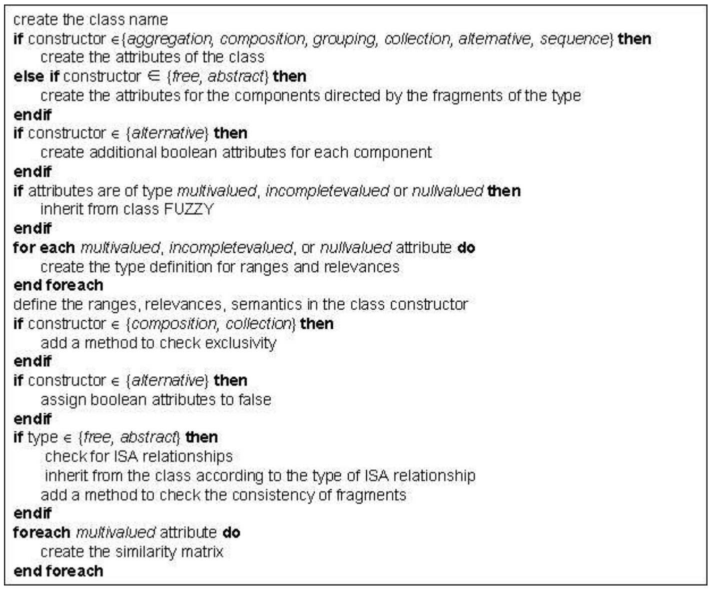
After building the conceptual data model, this model should be mapped to a logical model. In this case, the logical model is chosen as fuzzy object-oriented data (FOOD) model. The types, constructors, fragments, and ISA relationships that are used in the model should be appropriately transformed.

There are four kinds of building blocks of an ExIFO$_2$ schema: types, constructors, fragments and ISA relationships. *Aggregation, composition, grouping, collection, alternative* and *sequence* can be considered as complex constructors that have components. The types such as *printable, incomplete-valued, null-valued,* and *fuzzy-valued* are simple types and are components of other types or constructors. The *free* and *abstract* types might also have components through the fragments associated with them. The ISA relationships are essential only between *free* and *abstract* types. Components can be composed of constructors and types. Before going on details of transformation, an initial class FUZZY is created in FOOD. This class has attributes for membership and methods for determining class/subclass and class/object level uncertainty.

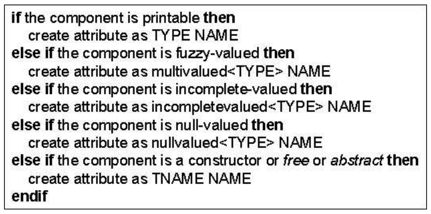
Before investigating the mapping process, it is better to revise the class structure. A class has a *name*, a list of *attributes* with their types and a list of *methods.* A class has also a list of classes that it inherits. A class is created for each of the constructors, *aggregation, composition, grouping, alternative,* and *sequence* constructors. A class is also created for *free* and *abstract* types. For constructors, attributes consist of its components. For *free* and *abstract* types, the attributes are composed of the elements directed by the fragments. The methods for the class can be functions to determine class/object level membership and class/subclass membership. There might be some additional methods to check the consistency of the constructors. For example, for composition constructor a class has a method to check exclusivity of its components. For the fragments of *free* and *abstract* types, methods to check the consistency of the fragments are added. The classes to inherit are determined by ISA relationships in case of *free* and *abstract* types. If a class includes an attribute derived from fuzzy-valued, incomplete-valued, and null-valued constructor, the class also inherits from class FUZZY. In case of *grouping, collection,* and *sequence* constructors, additional data structures might be needed to check the properties of their components such as *set*, *ordered set* properties.

The mapping is investigated in three parts. In the first part, the types that are components of other constructors or components directed by the fragments of *free* and *abstract* types are stated. These types are *printable, fuzzy-valued, incomplete-valued,* and *null-valued.* In the second part, the mapping of constructors are stated. In the third part, the transformation of *free* and *abstract* types is expressed briefly. The last part is isolated from the first part, since a class is created for these types. It is separated from the second part, since these types have fragments and might have ISA relationships associated with them.

The algorithm for generating classes for structures is presented in Figure 10 (a). The algorithm for the generation of attributes from components directed by *free* or *abstract* types is shown in Figure 10 (b). TYPE corresponds to the type name in FOOD of ExIFO$_2$ type and NAME represents the ExIFO$_2$ type name.



```
create the class name
if constructor ∈{aggregation, composition, grouping, collection, alternative, sequence} then
    create the attributes of the class
else if constructor ∈ {free, abstract} then
    create the attributes for the components directed by the fragments of the type
endif
if constructor ∈ {alternative} then
    create additional boolean attributes for each component
endif
if attributes are of type multivalued, incompletevalued or nullvalued then
    inherit from class FUZZY
endif
for each multivalued, incompletevalued, or nullvalued attribute do
    create the type definition for ranges and relevances
end foreach
define the ranges, relevances, semantics in the class constructor
if constructor ∈ {composition, collection} then
    add a method to check exclusivity
endif
if constructor ∈ {alternative} then
    assign boolean attributes to false
endif
if type ∈ {free, abstract} then
    check for ISA relationships
    inherit from the class according to the type of ISA relationship
    add a method to check the consistency of fragments
endif
foreach multivalued attribute do
    create the similarity matrix
end foreach
```



```
if the component is printable then
    create attribute as TYPE NAME
else if the component is fuzzy-valued then
    create attribute as multivalued<TYPE> NAME
else if the component is incomplete-valued then
    create attribute as incompletevalued<TYPE> NAME
else if the component is null-valued then
    create attribute as nullvalued<TYPE> NAME
else if the component is a constructor or free or abstract then
    create attribute as TNAME NAME
endif
```

Figure 10: Mapping algorithm for (a) constructors and *free* and *abstract* types (b) simple types.

6.1 Mapping of Types: Printable, Fuzzy-Valued, Incomplete-Valued and Null-Valued

These are separated from the others since these have simple structure and are transformed as attributes of classes. Each *printable* type has a name and a type name. For example, the *file format* in Figure 1 (a) is a type having name *fileformat* and has a type name *char **. This type is mapped to FOOD code as

*char * fileformat*

The type names can be types that previously exist such as *int, float*, etc. which correspond to integer and real values, respectively. They can also be new type names that are derived from previously existing type names such as *string*.

The types *fuzzy-valued, incomplete-valued,* and *null-valued* types have different structure since they might have more than one value and include uncertainty and fuzziness. Template classes are created for each of these types. These classes are *multivalued*, *incompletevalued* and *nullvalued* for fuzzy-valued, incomplete-valued and null-valued, respectively. For example, *when* in Figure 1 (b) is denoted as follows: if it has values of integers:

multivalued<int> when

The type *when* might have two types of values, integers and strings. Integer values correspond to the exact time where as string values denotes fuzzy time. Some of fuzzy values for *when* are late, early, etc. This type also supports semantics of the attribute. The *incompletevalued* has a range for its values. The *nullvalued* type allows additional values such as *unknown*, *does_not_exist*, and *no_information*.

6.2 Mapping of Constructors: Aggregation, Composition, Grouping, Collection, Alternative and Sequence

Each of these constructors has components. *Aggregation, composition* and *alternative* constructors have multiple components whereas the others have single components. For each of these constructors, a class is created. And the components are added to the class as attributes of the class. Class name is composed of letter "T" and the constructor name. The letter "T" represents that this class is used as a type. If the constructor is *alternative*, additional boolean attributes are defined to determine which component is the actual component. For composition and collection constructors methods for checking exclusivity of elements are added. Grouping and collection types represent set structure. A simple data structure can be built to represent sets. *Sequence* constructor can be considered as an ordered set. If a constructor has a fuzzy-valued, incomplete-valued or null-valued component, the generated class inherits from class FUZZY.

These constructors can also be components of other types. In these cases, their class name is associated in front of its name as its type when creating as an attribute.

6.3 Mapping of Types: Free and Abstract

The classes for these types are created as in case of *aggregation* constructor. The only difference is that the components which are directed by the fragments are added as the attributes of the class. The fragments can be *complex-total, total, complex-partial,* and *partial.* To preserve the consistency of the fragments, a function is added to the class. In addition, classes to inherit from exist and these are represented with ISA relationships. According to the ISA relation type (i.e., *generalization* or *specialization*) and the direction of the ISA relation, the class to inherit from is determined. The *free* and *abstract* types can also be components of other types. In these cases, their class name is associated in front of its name when creating as an attribute.

6.4 Creation of Ranges, Relevances, Semantics, and Similarity Matrices

If a class has attributes of type multivalued, nullvalued or incompletevalued attributes, then this class has ranges, relevances and semantics for these attributes. Similarity matrices for the elements in the ranges of multivalued attributes are also created. In this work relevances ranges, and semantics are declared in the class constructor. The similarity matrix is defined out of the class definition.

6.5 Sample Mappings from ExIFO$_2$ to FOOD

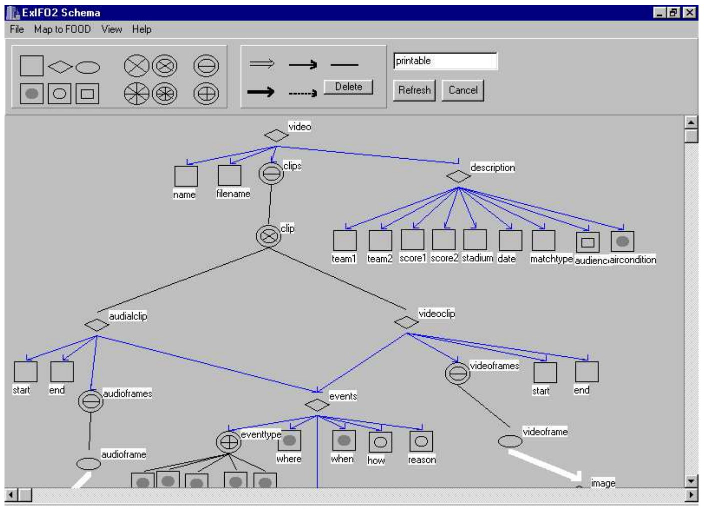
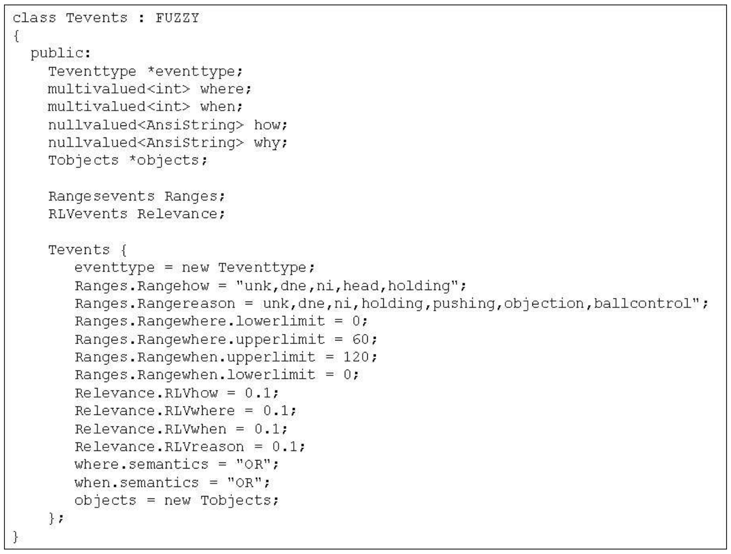


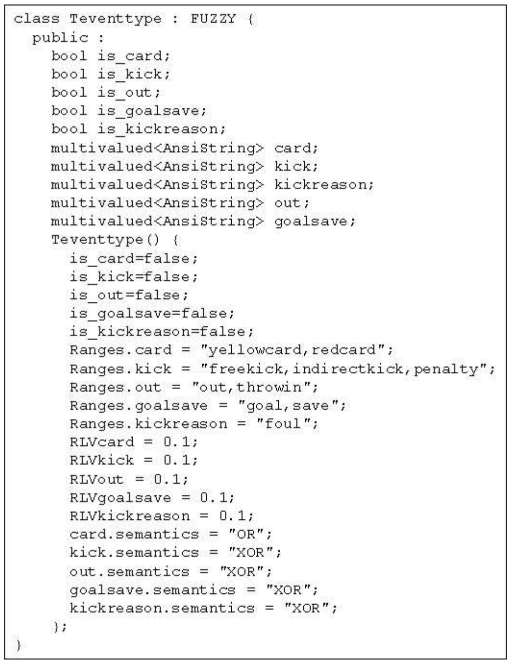
Figure 11: User Interface for Drawing ExIFO$_2$ Schema

In this section, we only show the mapping of some portions of the ExIFO$_2$ schema developed for videos of football games. The user interface for drawing ExIFO$_2$ schemas is shown in Figure 11. Firstly, the mapping of the event schema depicted in Figure 7 (b) will be given as an example. The FOOD code for this *abstract* type is shown in Figure 12 (a).

The class name is generated as *Tevents*. The attributes of the class are *eventtype, where, when, how, why* and *objects*. The types of *when* and *where* are defined as *multivalued<int>*. The values of *when* and *where* can be both crisp and fuzzy. The types of *how* and *why* are generated as *nullvalued<AnsiString>* since these attributes can take values corresponding to *no information, does not exist* and *unknown*. *AnsiString* is a builtin type for strings. Since *eventtype* is depicted with the alternative constructor, a new class will be generated for this attribute and its name will be *Teventtype*. Objects are denoted with the collection constructor, so its type will be its class name. There are two additional attributes related about the ranges and relevances of the class. The type definitions for ranges and relevances are generated and the attributes are shown in the constructor of the *Tevents* class. The type definitions will be stated later in this section. The corresponding



```
class Tevents : FUZZY
{
  public:
    Teventtype *eventtype;
    multivalued<int> where;
    multivalued<int> when;
    nullvalued<AnsiString> how;
    nullvalued<AnsiString> why;
    Tobjects *objects;

    Rangesevents Ranges;
    RLVevents Relevance;

    Tevents {
       eventtype = new Teventtype;
       Ranges.Rangehow = "unk,dne,ni,head,holding";
       Ranges.Rangereason = unk,dne,ni,holding,pushing,objection,ballcontrol";
       Ranges.Rangewhere.lowerlimit = 0;
       Ranges.Rangewhere.upperlimit = 60;
       Ranges.Rangewhen.upperlimit = 120;
       Ranges.Rangewhen.lowerlimit = 0;
       Relevance.RLVhow = 0.1;
       Relevance.RLVwhere = 0.1;
       Relevance.RLVwhen = 0.1;
       Relevance.RLVreason = 0.1;
       where.semantics = "OR";
       when.semantics = "OR";
       objects = new Tobjects;
    };
}
```



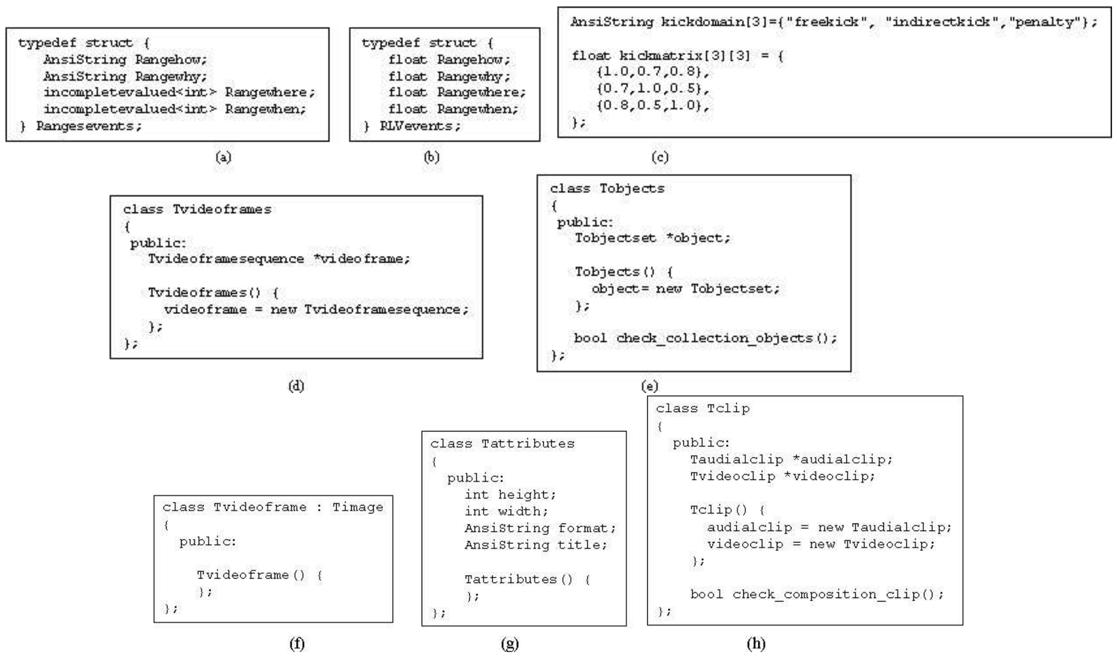
```
class Teventtype : FUZZY {
  public :
    bool is_card;
    bool is_kick;
    bool is_out;
    bool is_goalsave;
    bool is_kickreason;
    multivalued<AnsiString> card;
    multivalued<AnsiString> kick;
    multivalued<AnsiString> kickreason;
    multivalued<AnsiString> out;
    multivalued<AnsiString> goalsave;
    Teventtype() {
      is_card=false;
      is_kick=false;
      is_out=false;
      is_goalsave=false;
      is_kickreason=false;
      Ranges.card = "yellowcard,redcard";
      Ranges.kick = "freekick,indirectkick,penalty";
      Ranges.out = "out,throwin";
      Ranges.goalsave = "goal,save";
      Ranges.kickreason = "foul";
      RLVcard = 0.1;
      RLVkick = 0.1;
      RLVout = 0.1;
      RLVgoalsave = 0.1;
      RLVkickreason = 0.1;
      card.semantics = "OR";
      kick.semantics = "XOR";
      out.semantics = "XOR";
      goalsave.semantics = "XOR";
      kickreason.semantics = "XOR";
    };
}
```

(a) (b)

Figure 12: Codes for mapping of (a) *abstract* type (b) *alternative* constructor.

ranges and relevances are assigned to these attributes. The semantics of the attributes *where* and *when* are also assigned in the constructor. The assignments are done in the constructor of the class since these will be initial values for all of the objects. Instead, methods could be defined for assignments of ranges, relevances, and semantics, but in this case the user should call these methods after creation of the object. Since each event will have an event type and a set of objects, these are created with the *new* command in the constructor. As the last point for this class declaration, this class inherits from class FUZZY since it includes multivalued and nullvalued attributes.

Figure 12 (b) shows the FOOD code for the alternative constructor that is used for event type. In this case, a boolean attribute for each component to determine which one of the event types is true is added to the declaration. Alternative constructor requires only one of the components to be true, i.e., they are mutually exclusive. In the constructor of the class, these are initially assigned to *false* since none of the events is chosen when the object is created. When the event type is defined, the corresponding boolean attribute will be transformed to *true*. The other attributes of the class are declared as *multivalued<AnsiString>*. The values of ranges, relevances and semantics are assigned in the constructor of the class. Assigning equal relevance values means that each component has the same effect on determining membership of the object to the class.



```
typedef struct {
   AnsiString Rangehow;
   AnsiString Rangewhy;
   incompletevalued<int> Rangewhere;
   incompletevalued<int> Rangewhen;
} Rangesevents;
```

(a)

```
typedef struct {
   float Rangehow;
   float Rangewhy;
   float Rangewhere;
   float Rangewhen;
} RLVevents;
```

(b)

```
AnsiString kickdomain[3]={"freekick", "indirectkick","penalty"};

float kickmatrix[3][3] = {
   {1.0,0.7,0.8},
   {0.7,1.0,0.5},
   {0.8,0.5,1.0},
};
```

(c)

```
class Tvideoframes
{
 public:
   Tvideoframesequence *videoframe;

   Tvideoframes() {
     videoframe = new Tvideoframesequence;
   };
};
```

(d)

```
class Tobjects
{
 public:
   Tobjectset *object;

   Tobjects() {
     object= new Tobjectset;
   };

   bool check_collection_objects();
};
```

(e)

```
class Tvideoframe : Timage
{
  public:

    Tvideoframe() {
    };
};
```

(f)

```
class Tattributes
{
  public:
    int height;
    int width;
    AnsiString format;
    AnsiString title;

    Tattributes() {
    };
};
```

(g)

```
class Tclip
{
  public:
    Taudialclip *audialclip;
    Tvideoclip *videoclip;

    Tclip() {
      audialclip = new Taudialclip;
      videoclip = new Tvideoclip;
    };

    bool check_composition_clip();
};
```

(h)

Figure 13: Mapping to FOOD (a) ranges (b) relevances (c) similarity matrix (d) *sequence* constructor (e) *collection* constructor (f) *specialization* (g) *aggregation* constructor (h) *composition* constructor.

Figure 13 depicts mappings for *ranges, relevances, similarity matrix, sequence, collection, specialization, aggregation,* and *composition.*

7 Querying

The index structures have been built for events and the objects in the clips. The data structure used for accessing the objects about the clips in which they appear is a simple sorted list of the objects. Each object name is associated with the clips that they exist along with their roles. When a user specifies a query, the system first checks if the object exists. If it exists, the clips that the objects are seen are retrieved. Additionally, if the role of the object is set front as a constraint, the clips are controlled along with this restriction.

A simple data structure is built to perform fast retrieval of events from the database. This is a tree structure in which at the first level video names are branched. Then each video has five branches where each branch corresponds to an event type. The events are ordered according to time that they occur and have pointers to clips. When a user submits a query, the clips are firstly retrieved from this tree with respect to event types and event times. This operation reduces the number of clips to be checked at the first attempt. This structure also enables the fast retrieval of events defined with a fuzzy time value as well as crisp value. When a user submits a query including a fuzzy term with a threshold for the time of the event, firstly a function takes the fuzzy term and threshold as input and returns the range of time to be retrieved as output. The events which lay in this range are retrieved. In this application, we only performed indexing on fuzzy time as an example for fuzzy terms but this can be extended also for other fuzzy terms.

The query construction has 3 components. It is composed of the selection of properties of video, the selection of events and the selection of objects along with their properties. The querying process is composed of the following steps:

- specification of video description properties(optional)
- specification of event properties (optional)
- specification of object properties (optional)
- retrieving clips by using the index structures
- comparing the information in clips with the rest of the attributes and elimination of non-relevant clips
- presentation of query results

7.1 Query Construction

7.1.1 Querying for Video Description

At the first level of querying, the user can set up queries about video description. As mentioned before, the video description consists of team names, team scores, air type, audience number, game type, stadium, and date. The user might request videos of games which are played in stadium "Parc de Prince" or played in "June 20th of 1998". The user can also determine the game type. He can request the videos of "World-Cup" or "friendly" games. The user might want to watch games played in a "rainy" or "sunny" weather. Since air condition is a fuzzy term, the user can state a threshold for similarity. He might also be interested in the number of audience. As stated before, the audience number lays in a range. The user can define the range of audience along with a threshold. These are the opportunities given to the user for building queries. The user is usually interested in the team names and team scores. The user can request games of England. In addition, he can put a restriction that England plays against Germany. Besides stating the visitor team, he can also specify a score and request games which England played against Germany and scored 3. If the user is interested in number of goals of games, he might request games according to goal count in the game. He can request "all videos that many goals occur" or "all videos that 2 goals occur".

7.1.2 Querying for Event

After the user emphasizes restrictions about video description, the videos that do not fit user's declaration are directly eliminated by the above processing, and queries including further details about events and objects are processed with the rest. For the videos retrieved, the event and object information are controlled. Different indexing structures are kept for events and objects. Only one of the structures is traversed when a query is submitted including object information or event information. It is obvious that when information only about events is stated, then event index structure is traversed and when information only about objects is stated, then object index structure is traversed for the corresponding clips. When information for both of them is included, it is burdensome to traverse both index structures, hence only one of them is searched. In this application, the index structure that is traversed is chosen as the event index structure when information is given both about events and objects.

After the number of videos to be retrieved is minimized, querying about events and objects can be performed. In this application, the user can determine the events that he wants to watch. The event types are fuzzy terms. Therefore, event types can have a threshold. As notified before, each clip consists of events. The user's focus is usually the clips that contain these events. The user might request the *goals* or *fouls* of a game. Sometimes, the user might not be sure about the event name such as *yellow card* or *red card* and he can state the event name with a comma such as *yellowcard, redcard.* In these cases, the event type is extracted from the event name firstly. In this occasion, the event type is *card* event. Since the semantics for *card* event is OR, this event name would have been searched with conjunction along with the threshold. If the user had stated the event name as *free kick, indirect kick* pair, the event type would be *kick* event. Since *kick* event has XOR semantics, event name is searched according to XOR semantics with the corresponding threshold. The user does not need to specify the event type semantics for event names. These are extracted automatically by the program.

In the above case, only one event is defined. Clips contain at least one event, and the user could define complex queries for clips. He can request "clips that contain *fouls* and *yellow card, red card*". The user can define also *how*, *why* the event happened. "Retrieve clips that contain fouls caused by holding the player" is a sample query including *how* attribute. The time and the location of the event are important in football games. The user can crisply state the event time such as 75th minute. He can also state the event time using fuzzy terms. There are 6 fuzzy terms defined for time, which are *at the beginning of the game, at the mid of the first-half, at the end of the first half, at the beginning of the second-half, at the mid of the second-half*, and *at the end of the game.* The user can request "the goals which are scored at 85th minute". The user can also request "the goals scored towards the end of the game" with a threshold. In this application, he can set up queries including fuzzy terms and crisp terms for the same attribute such as *time.* The location can also be defined crisply or using fuzzy terms. The user can request goals scored with a distance of 2 meters to the goalpost. He can also set up a query "retrieve all goals scored near the goalpost". "Retrieve all goals scored near the goalpost from free kick towards the end of the game" is a sample complex query for an event.

7.1.3 Querying for Object

Besides specifying constraints about events, the user can also declare object properties. Objects have roles and relations with other objects in a clip. A player can be a scorer in one event or he can be a performer of a free kick event in another event. In this sense, "retrieve all free kicks performed by Shearer" or "retrieve all goals scored by Shearer" are sample queries. One of the important objects in a game is the *ball.* The position of the *ball* with respect to goalpost when a player scores might be important. This kind of relation is a *spatial* relation. There are 9 relation names for the ball with respect to goalpost. These are *left-bottom, middle-bottom, right-bottom, left-middle, middle-middle, right-middle, left-top, middle-top*, and *right-top.* These locations are shown in Figure 14. Each of these relations has some degree of similarity among them. These relation names can be stated with a threshold. "Retrieve all goals scored to the left-top(0.8) of the goalpost" is a sample query. The players can also have spatial relationships with other players such as *left-of* and *right of.* "Retrieve all goals by Shearer who scored the ball to the left-top of the goalpost with his head towards the end of the game near the goalpost" is a complex query for an object.

The properties about the players can also be stated. The player has a team name, height, age

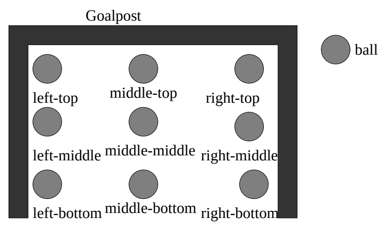


Figure 14: Spatial Relation

```
select clips from football_database
where
      description.teamname1 = "England" and description.teamscore1 = 2 and description.teamname2 = "Germany"
   and description.matchtype = "World-Cup" and description.stadium = "Old Trafford" and description.audience = 50000-60000(0.8)
   and description.airtype = "sunny"(0.8) and description.goalcount = "many"(0.8) and event.name = "goal"
   and event.when="at the end of match"(0.9) and event.how = "head" and event.where = "near" and event.object1.name = "ball"
   and event.object[1].relation = "left-top"(0.9) and event.object1.relatedobject = "goalbar" and event.object[2].role = "scorer"
   and event.object[2].age = "young"(0.8) and event.object[2].height = "short"(0.8) and event.object[2].playertype = "middle-field"(0.8)
```

Figure 15: Sample Query

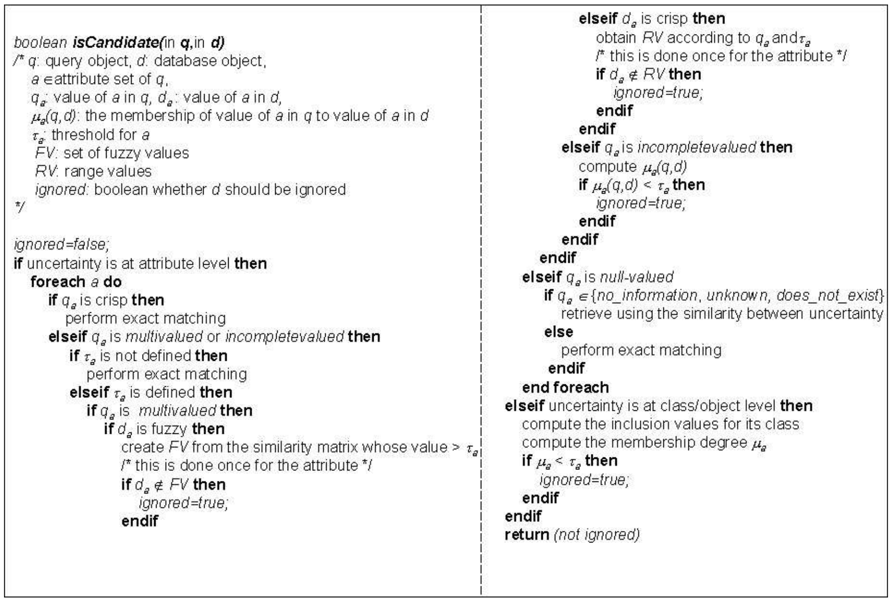
and player property. Height and age are fuzzy attributes. The player type has object/class level of uncertainty. As mentioned before, the player can be a middle-field player and a forward player with different degrees of membership. "Retrieve all goals scored by young, tall, defender players" is a sample query about player properties. *Young, tall,* and *defender* are fuzzy values. A very simple query about a player is "retrieve all clips in which Shearer exists".

7.1.4 Querying for Event & Object

Finally, the following is a combination of the previously stated attributes that a user can specify: "Retrieve all clips in which a young(0.8), short(0.8), and a middle-field(0.8) player scores the ball to the left-top(0.9) of the goalpost with his head towards the end of the game(0.9) near the goal-post in videos in which England scored 2 against Germany in a World Cup which 50000-60000(0.8) audience watched the game in stadium "Old Trafford" in a sunny(0.8) day and where many(0.8) goals are scored". This query can be expressed more formally as in Figure 15. The numbers in the parenthesises next to fuzzy values are thresholds for the fuzzy attributes. If threshold is not defined, it is assumed 1. If no condition had been specified, then all clips from selected videos would have been retrieved.

7.2 Querying Process

The next step after the specification of information about events and objects, is the processing of attributes of objects and events. The core part in the querying process is the comparison of the attributes. Exact matching of crisp values either corresponds to checking the equality of values or checking the membership of value to a class. If an attribute in the database object is fuzzy and the corresponding attribute in the query is multivalued, a set of fuzzy values is created using the similarity matrix and the threshold for the attribute. If the database object has a crisp value and the corresponding attribute in the query is multivalued, the range values are computed for the attribute. Both of these computations are done once for a query. The membership value of the query object to the database object needs to be computed if an attribute in the query is incomplete-valued. If the uncertainty is at class/object level, the membership of the query object to the class is computed. Null-valued attributes are compared using the similarity matrix for null-valued attributes. The detailed algorithm for comparisons of attributes is given in Figure 16.



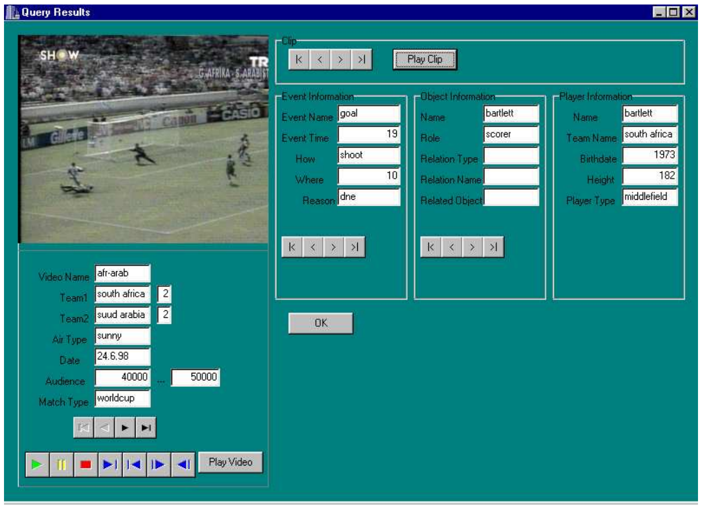
```
boolean isCandidate(in q,in d)
/* q: query object, d: database object,
   a ∈attribute set of q,
   q_a: value of a in q, d_a: value of a in d,
   μ_a(q,d): the membership of value of a in q to value of a in d
   τ_a: threshold for a
    FV: set of fuzzy values
    RV: range values
    ignored: boolean whether d should be ignored
*/

ignored=false;
if uncertainty is at attribute level then
   foreach a do
      if q_a is crisp then
         perform exact matching
      elseif q_a is multivalued or incompletevalued then
         if τ_a is not defined then
            perform exact matching
         elseif τ_a is defined then
            if q_a is  multivalued then
               if d_a is fuzzy then
                  create FV from the similarity matrix whose value > τ_a
                  /* this is done once for the attribute */
                  if d_a ∉ FV then
                     ignored=true;
                  endif
               elseif d_a is crisp then
                  obtain RV according to q_a and τ_a
                  /* this is done once for the attribute */
                  if d_a ∉ RV then
                     ignored=true;
                  endif
               endif
            elseif q_a is incompletevalued then
               compute μ_a(q,d)
               if μ_a(q,d) < τ_a then
                  ignored=true;
               endif
            endif
         endif
      elseif q_a is null-valued
         if q_a ∈{no_information, unknown, does_not_exist}
            retrieve using the similarity between uncertainty
         else
            perform exact matching
         endif
   end foreach
elseif uncertainty is at class/object level then
   compute the inclusion values for its class
   compute the membership degree μ_a
   if μ_a < τ_a then
      ignored=true;
   endif
endif
return (not ignored)
```

Figure 16: Algorithm for comparison of imprecise values.

7.3 Experiments

After the user submits the query, the results of the query are presented to the user. The user can see the information about the video, clips, events, and objects. The user can navigate through the videos, the clips in a video, the events in a clip or the event objects. The user can play a clip by selecting the clip or can play all the video (Figure 17).

The database update and insertion are more convenient than in traditional databases. The main reason is that the user can enter data at any level of uncertainty. Whatever information that



Video Name: afr-arab
Team1: south africa 2
Team2: suud arabia 2
Air Type: sunny
Date: 24.6.98
Audience: 40000 ... 50000
Match Type: worldcup

Event Name: goal
Event Time: 19
How: shoot
Where: 10
Reason: dne

Name: bartlett
Role: scorer
Relation Type:
Relation Name:
Related Object:

Name: bartlett
Team Name: south africa
Birthdate: 1973
Height: 182
Player Type: middlefield

Figure 17: Query Results

is available at the moment is entered and used in retrieval. For example, the kick events are entered using XOR semantics when the kick event is not exactly known. The card events are entered with OR semantics when the user is not sure of the card event. The distance information can be entered with fuzzy terms without knowing the exact distance values. In a traditional database, the user has to guess the distance value. Since the guessed distance is less likely to be the exact value, wrong information is kept in a traditional database. The creation of our database is easier since the user did not have to have the all exact information at the moment. The uncertain, imprecise, or fuzzy attributes can be converted to crisp values when the exact values are ready.

The users query the database easier than that in traditional databases. Since they have no idea on how the data is stored or actually what the type of data is, they are able to perform queries using fuzzy terms. For a football game, only experts know the correct measures of the field. Since users do not have idea of the actual distances of the field, it is hard for them to construct proper queries. In a traditional database, the user would perform a range query for a distance. Since the user does not need to know the actual distances in the field, he would perform a couple of queries until he actually receives what he needs. In our prototype, we have fuzzy terms for distances like very-close, close, far, and very-far. The user can use any of these fuzzy values without actually knowing the actual distances in the field. The user does not have to worry about range queries either since these parts are automatically performed by the query processor using fuzzy values.

The database consists of around 40 world cup and friendly games among national teams. The performance of the prototype is not degraded by the fuzzy information. This has been detected by performing separate queries for crisp values and fuzzy values. From the perspective of the user, the system responded these queries at similar times. The major difference from traditional query processing arises from the comparison of the attributes. For a fuzzy attribute, either the query or the database object may contain fuzzy values. Whenever an attribute has a fuzzy value, a set of fuzzy values are generated using the similarity matrix. Then fuzzy value is checked whether it is in the set. If the value is crisp then range values are computed for the attribute. The values are checked whether they are in the set or in the range. Both of the set and range computations are done once for a query. Similar computation for incomplete-valued values is performed. Therefore, the difference between traditional comparison and this one is that comparison of attributes having fuzzy, multivalued or incomplete-valued attributes turns into range or set containment check. For class/subclass level uncertainty, for each probable object (i.e., where the rest of the attributes fit to the query), the membership to the class has to be computed. This membership computation can be performed once and stored in the database. Therefore there will not be an additional computation for class/subclass level uncertainty.

8 Conclusion

In this work, we presented an approach for designing multimedia database applications at the conceptual level and the logical level. This work has three components. At the high level, the conceptual modeling of multimedia database applications using ExIFO$_2$ data model is performed. At the next level, this conceptual model is mapped to FOOD data model. At the low level, queries and retrieval based on FOOD model are done. applications include The user can request retrieval of objects having fuzzy, uncertain values as well as the crisp values for the same attribute. The user can navigate through the retrieved objects and can watch them if he desires. In our work, we modeled the media where the multimedia objects are presented and the multimedia objects themselves. We also modeled the multimedia objects and the relations between them. The ExIFO$_2$ model presents relationships of ISA type by *generalization* and *specialization* links and PART_OF relationships by complex constructors such as *grouping* and *aggregation*. The user subjectivity is minimized by keeping similarity matrices for fuzzy attributes. We can perform queries about spatial relationships among objects. These queries include fuzzy values. A detailed conceptual modeling of imprecise data for multimedia database applications has been provided in this report. This model is mapped into a logical model (FOOD) to show the applicability and the power of the model. The user is allowed to submit queries having values at different levels of uncertainty. In this work, the queries on temporal attributes such as time can be submitted. These queries include both uncertain and crisp values. For future work, the temporal relationships such as *before, after* between the objects should also be performed by comparison of clip starting and ending frames. In this report, our goal is to show incorporation and management of fuzzy data in multimedia database applications. Although the performance in our prototype is sufficient, for large databases efficient indexing structures should be used to increase the performance.

References

[1] S. Abiteboul, R. Hull, "IFO: A formal semantic database model," ACM Transactions on Database Systems, Vol.12, pp. 525-565, 1987

[2] S. Adali, S. Candan, S. Chen, K. Erol, S. Subrahmanian, "The advanced video information systems: data structures and query processing," Multimedia Systems, Vol.4, pp. 172-186, 1996

[3] M. Adiba, "STORM: an object-oriented multimedia DBMS," Multimedia Database Systems, pp. 47-88, 1996.

[4] S. Aygun, A. Yazici, and N. Arica, "Conceptual Data Modeling of Multimedia Database Applications", The proceedings of the 4th International Workshop on Multi-Media Database Management Systems, pp. 182-189, Dayton, Ohio, August 1998

[5] M. Bouzeghoub, E. Mtais, "Semantic modeling of object-oriented databases," In Proc. VLDB, pp. 3-14, 1991

[6] S-C. Chen, R. L. Kashyap, "A spatio-temporal semantic model for multimedia database systems and multimedia information systems," IEEE Trans. on Knowledge and Data Engineering, Vol. 13, No. 4, July/August 2001

[7] Y. F. Day, S. Dagtas, M. Iino, A. Khokhar, A. Ghafoor. "Object-oriented conceptual modeling of video data," 11th Conf. on Data Engineering, Taiwan, March, 1995, pp:401-408

[8] R. Fagin, "Fuzzy queries in multimedia database systems," Proc. 1998 ACM Symposium on Principles of Database Systems, 1998 pp. 1-10

[9] A. Ghafoor, "Special Issue on Multimedia Database Systems," Multimedia Systems, Vol. 3(5-6), pp. 179-181, 1995

[10] S. Gibbs, C. Breiteneder, D. Tsichritzis, "Audio/Video databases: an object-oriented approach", ICDE, 1993, pp. 381-390.

[11] V. Gudivada, V. Raghavan, "Modeling and retrieving images by content," Information Processing & Processing. Vol. 333, No.4, pp. 427-452, 1997

[12] P. Loucoupolous, R. Zicari, Conceptual modeling, databases and CASE: an integrated view of information systems development, Wiley Professional Computing, 1992.

[13] A. D. Narasimhalu, "Multimedia databases," Multimedia Systems, Vol. 4, pp. 226-249, 1996

[14] E. Oomoto, K. Tanaka, "OVID: design and implementation of a video object database system," IEEE Transactions on Knowledge Data Eng., 1993, Vol. 5, pp. 629-643

[15] F. Petry, Fuzzy Databases, Principles and Applications, Kluwer Academic Publishers, Boston (USA), 1996

[16] G. Schloss, M. Wynblatt, "Providing definition and temporal structure for multimedia data," Multimedia Systems, Vol. 3, pp. 264-277, 1995

[17] C. Sernadas, J. Fiadeiro, "Towards object-oriented conceptual modeling," Data & Knowledge Engineering, Vol. 6, pp. 479-508, 1991

[18] M. Teisseire, P. Poncelet, R. Cicchetti, "Towards event-driven modelling for database design," In Proc. of the 20th VLDB Conference, Santiago, Chile, 1994, pp. 285-196

[19] M. Vazirgiannis, "Multimedia Data Base Object and Application Modeling Issues and an Object-Oriented Model," in the book "Multimedia Database Systems: Design and Implementation Strategies " (editors Kingsley C. Nwosu, Bhavani Thuraisingham and P. Bruce Berra), Kluwer Academic Publishers, 1996, pp. 208-250

[20] L. Yang, J. Wu, “Towards a semantic image database system,” Data & Knowledge Engineering, Vol. 22, pp. 207-227, 1997

[21] Yazici, A., B. P.Buckles, F.E. Petry, ”Handling Complex and Uncertain Information in the ExIFO and NF2 Data Models,” IEEE Trans. on Fuzzy Systems, Vol. 7, No. 6, December 1999

[22] A. Yazici, 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 and New York, Vol 39., pp: 12-35, 2000

[23] A. Yazici, R. George, Fuzzy Database Modeling, Physica-Verlag, Heidelberg, 1999

[24] A. Yazici, R. George, D. Aksoy. Design and Implementation Issues in the Fuzzy Object-Oriented Data Model, Information Sciences Vol. 108/1-4, pp. 241-260, 1998

[25] L. A. Zadeh, “Similarity relations and fuzzy orderings,” Information Sciences, Vol.3, No.2: 177-200, 1971