Graph-Based Object-Oriented Approach for Structural and Behavioral Representation of Multimedia Data

Ivan Radev, *Senior Member, IEEE*
Department of
Computer Science
Merrimack College
North Andover, MA 01845
978-837-5000
iradev@usa.net

Niki Pissinou, Kia Makki
The Center For Advanced
Computer Studies
University of
Southwestern Louisiana
Lafayette, LA 70504
318-482-6604, 318-482-5872
pissinou@cacs.usl.edu
kia@usl.edu

E. K. Park
Computer Science
Telecommunications
University of Missouri
at Kansas City
Kansas City, MO 64110
816-235-1497
ekpark@cstp.umkc.edu

Abstract

The management of multimedia information poses special requirements for multimedia information systems. Both representation and retrieval of the complex and multifaceted multimedia data are not easily handled with the flat relational model and require new data models. In the last several years, object-oriented and graph-based data models are actively pursued approaches for handling the multimedia information. In this paper the characteristics of the novel graph-based object-oriented data model are presented. This model represents the structural and behavioral aspects of data that form multimedia information systems. It also provides for handling the continuously changing user requirements and the complexity of the schema and data representation in multimedia information systems using the schema versioning approach and perspective version abstraction.

1 Introduction

The management of multimedia information poses special requirements for multimedia information systems (MMISs). Both representation and retrieval of the complex and multifaceted multimedia data are not easily handled with the flat relational model and require new data models. The object-oriented data models provide powerful abstraction and structural and behavioral mechanisms to represent the complex multimedia information specifying the database schemas. On the other hand, MMISs are much more prone to interactions with the users, especially during the processes of information navigation and retrieval. Graph-based data models are particularly appropriate for handling the continuously changing user requirements during such interactions. Most of the graph-based models [4, 5] only provide the structural modeling of the information using graph-representation of database schema, and none of them provides efficient behavioral manipulation of the multimedia information to the best of our knowledge. However, the complexity of the multimedia information requires a versatile and robust way to determine its behavior defining operations on it.

The complexity of the information on schema and instances levels requires applying appropriate manners to reduce it to a manageable level for the user. Here, schema evolution techniques based on schema versioning approach [3] are actively investigated area.

In the present paper, the characteristics of the novel graph-based object-oriented data model (GBOODM) are presented. This model represents the structural and behavioral aspects of data that form multimedia information systems. It also provides for handling the continuously changing user requirements and the complexity of the schema and data representation in MMISs using the schema versioning approach and perspective version abstraction (discussed in Section 3).

The paper is organized as follows. In Section 2, the structure and behavior of the GBOODM model is presented. Then, in Section 3, the schema versioning part of the model is discussed. Finally, a conclusion and future work is indicated.

2 Structure and Behavior of the Model

The representation of the model includes conceptual schema and its corresponding conceptual schema graph, which both define the intensional level of a database. It also includes the information system and its corresponding instance graph, which, in turn, both define the extensional level of the database.

2.1 Conceptual Schema—The Intensional Level

The *conceptual schema* of the model is defined as an eight tuple abstraction

$$\Sigma = (C, T, A, Op, \mathcal{P}, \mathcal{O}, \mathcal{H}, \mathcal{U})$$

The tuple elements have the following meaning:

- C is a finite set of class names, where each class $c \in C$ denotes a structure (through its attributes) with a behavior (through its operations) on an intensional level and a set of objects with that structure and behavior on an extensional level.
- T is a finite set of built-in type names, where each $t \in T$ denotes a primitive object type and $V(T)$ is the set of associated values, which belong to this type.
- A is a finite set of attribute names, defined on classes and representing the structure of the corresponding class instances (objects). Attributes are divided into simple, having the domain $t \in T$, and complex, having the domain $c \in C$. They are also divided on single-valued attributes A_s and multivalued attributes A_m, where $A = A_s \cup A_m$.
- Op is a finite set of operation names belonging to classes and representing the behavior of the corresponding class instances. An operation can have parameters and can return a result that can be of a basic type $t \in T$ or of a class $c \in C$.
- $\mathcal{P} \subseteq C \times A \times (C \cup T)$ is the property relationship. If (c_i, a, c_j) or $(c_i, a, t_j) \in \mathcal{P}$, then the class c_i has the attribute a with the domain c_j or t_j.
- $\mathcal{O} \subseteq \mathcal{OPS} \times \mathcal{IN} \times \mathcal{OUT}$ is the operation relationship signature, where:
  - $\mathcal{OPS} \subseteq C \times Op$ is the operation participating relationship signature, and if $(c_i, op_i) \in \mathcal{OPS}$, then the operation instance op_i participates in the behavior declaration (signature) of class c_i.
  - $\mathcal{IN} \subseteq (C \cup T) \times A \times Op$ is the operation input relationship, and if $(c_j, a_{i_k}, op_i) \in \mathcal{IN}$ or $(t_j, a_{i_k}, op_i) \in \mathcal{IN}$, then the operation op_i has as an input formal parameter the attribute a_{i_k} with the domain c_j or t_j, respectively.
  - $\mathcal{OUT} \subseteq Op \times A \times (C \cup T)$ is the operation output relationship, and if $(op_i, a_{o_k}, c_l) \in \mathcal{OUT}$ or $(op_i, a_{o_k}, t_l) \in \mathcal{OUT}$, then the operation op_i has as a result the attribute a_{o_k} with the domain c_l or t_l, respectively.
- $\mathcal{H} \subseteq C \times C \times C \times \cdots \times C$ is the multiple inheritance relationship defining partial order among classes. If $(c_i, c_{j_1}, c_{j_2}, \ldots, c_{j_n}) \in \mathcal{H}$, then the class c_i is a subclass of $c_{j_1}, c_{j_2}, \ldots, c_{j_n}$, in which the attributes and operations are inherited from its superclasses $c_{j_1}, c_{j_2}, \ldots, c_{j_n}$. By default, the operations are replicated but they can be explicitly overridden. and the attributes can be modified. Additionally, new attributes and operations specific to the subclass c_i can be defined. The name conflicts among superclasses of a given class due to the multiple inheritance are resolved by choosing the name of an attribute or an operation which has the same name in several superclasses to be the name belonging to the first immediate superclass from left to right in the list of superclasses.
- $\mathcal{U}$ is a schema versioning tuple containing the schema update primitives (SUPs). SUPs are used to define the changes over the source schema Σ when a new schema version of Σ, Σ' is defined. The tuple $\mathcal{U}$ is defined as follows:

  $\mathcal{U} = (AEA, AED, AERN, AERT, MNA, MND, MNRN, MNSC, MNCDC, IEA, IED, CNA, CND, CNRN)$

  The elements of the tuple have the following meaning:

  AEA—Attribute Edge Addition

  AED—Attribute Edge Deletion

  AERN—Attribute Edge Rename

  AERT—Attribute Edge Retype

  MNA—Method Node Addition

  MND—Method Node Deletion

  MNRN—Method Node Rename

  MNSC—Method Node Signature Change

  MNCDC—Method Node Code Change

  IEA—Inheritance Edge Addition

  IED—Inheritance Edge Deletion

  CNA—Class Node Addition

  CND—Class Node Deletion

  CNRN—Class Node Rename

  The formal definitions of some of the schema update primitives are given in the next section.

The *conceptual schema graph*, $G(\Sigma)$, corresponding to the conceptual schema, Σ, of the graph-based object-oriented data model is represented through a direct labeled graph in which the nodes N_Σ and edges E_Σ are determined as follows:

- $N_\Sigma = C \cup T \cup Op$ is the set of nodes. The classes $c \in C$ are represented by rectangle with label c, the types $t \in T$ are represented by oval-shaped nodes with label t and the operations $op \in Op$ are represented by circles with label op.

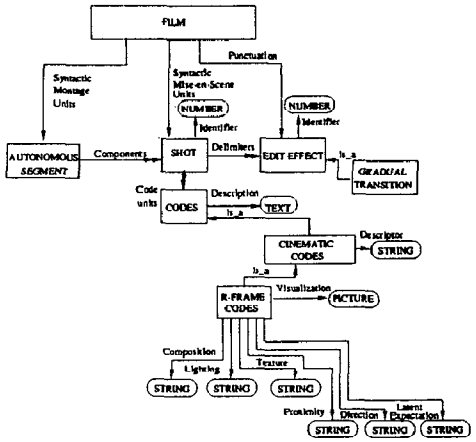

Figure 1: Conceptual Schema Graph

- E_Σ is the set of edges. Each $(c_i, a, c_j) \in \mathcal{P}$ is represented by a single-lined edge from c_i to c_j with label a, and if $a \in A_s$, then the edge is single- arrowed ($c_i \overset{a}{\rightarrow} c_j$), while if $a \in A_m$, then the edge is double-arrowed ($c_i \overset{a}{\rightarrow\!\!>} c_j$). Each $(c_i, op_i) \in \mathcal{OPS}$ is represented by a double-lined edge from c_i to op_i ($c_i \Longrightarrow op_i$). Each $(c_j, a_{i_k}, op_i) \in \mathcal{IN}$ or $(t_j, a_{i_k}, op_i) \in \mathcal{IN}$ is represented by a single-lined dotted edge from c_j or t_j to op_i with the label a_{i_k} (e.g., $c_j \overset{a_{i_k}}{\cdots\!>} op_i$). Each $(op_i, a_{o_k}, c_l) \in \mathcal{OUT}$ or $(op_i, a_{o_k}, t_l) \in \mathcal{OUT}$ is represented by a single-lined dotted edge from op_i to c_l or t_l with the label a_{o_k} (e.g., $op_i \overset{a_{o_k}}{\cdots\!>} c_l$). Finally, each $(c_i, c_j) \in \mathcal{H}$ is represented by a single-lined bold edge from c_i to c_j with the label "is-a" ($c_i \overset{\text{is-a}}{-\!>} c_j$).

The conceptual schema graph $G(\Sigma)$ carries the relevant information which is necessary to determine Σ. An example of a fragment of a conceptual schema graph is shown in Figure 1, where the use of class and attribute nodes, as well as of attribute and inheritance edges is illustrated. The example represents part of the syntactic modeling of film video, in correspondence to which the film video annotation is done. Video annotation is a process in which video characteristics are extracted, represented and organized guided by a model of the video application domain. In this paper, the application domain (film video) details are not discussed. These are considered in another paper [6]. This example of a conceptual schema graph will be used in the rest of this paper to illustrate the capabilities of the GBOODM model.

2.2 Information System— The Extensional Level

Given the schema of the graph-based object-oriented data model defined above, the *information system, IS,* representing the corresponding extensional (instance) level is defined as a six tuple abstraction

$$IS = (\Sigma, O, Mt, \mathcal{J}, \mathcal{L}, \mathcal{BL})$$

The tuple elements have the following meaning:

- Σ is the conceptual schema of the model.
- O is the set of objects in the information system IS, where $o_j \in O, j = (1, n_{o_j})$ and n_{o_j} is the number of objects o_j (cardinality of O) participating in the IS.
- Mt is the set of methods in the information system IS.
- $\mathcal{J} \subseteq O \times C$ is the instantiation relationship. Each $o_j \in O$ is an instance of $c_j \in C$.
- $\mathcal{L} \subseteq O \times A \times (O \cup V(T))$ is the link relationship. If $(o_i, a, o_j) \in \mathcal{L}$ then the attribute a of the object o_i has the value o_j (Figure 2a). In the case of a single inheritance, where a class inherits the attributes of its single superclass, if o_i and o_j are instances of c_i and c_j, respectively, then $(o_i, a, o_j) \in \mathcal{L}$[1] is a valid link relationship iff one of the following cases is valid:
  - *new defined attribute*, $(c_i, a, c_j) \in \mathcal{P}$[2] (Figure 2b)
  - *inherited attribute*, $(c_i, c_k) \in \mathcal{H}$ and $(c_k, a, c_j) \in \mathcal{P}$ (Figure 2c)
  - *subclass in a place of superclass attribute*, $(c_j, c_k) \in \mathcal{H}$ and $(c_i, a, c_k) \in \mathcal{P}$ (Figure 2d).

  Generalizing the case of a single inheritance to a multiple inheritance, where a class inherits attributes from its several superclasses, if o_i, o_{j_1}, o_{j_2}, ..., o_{j_n} are instances of c_i, c_{j_1}, c_{j_2}, ..., c_{j_n}, respectively, then $(o_i, a_1, o_{j_1}) \in \mathcal{L}$, $(o_i, a_2, o_{j_2}) \in \mathcal{L}$, ..., $(o_i, a_n, o_{j_n}) \in \mathcal{L}$ (Figure 3a) are valid link relationships iff the following is valid:
  - *inherited attributes*, $(c_i, c_{k_1}, c_{k_2}, \ldots, c_{k_n}) \in \mathcal{H}$, $(c_{k_1}, a_1, c_{j_1}) \in \mathcal{P}$, $(c_{k_2}, a_2, c_{j_2}) \in \mathcal{P}, \ldots, (c_{k_n}, a_n, c_{j_n}) \in \mathcal{P}$, and $a_1 \neq a_2 \neq \cdots \neq a_n$, where n is the number of the inherited attributes (Figure 3b). Note that in Figure 3a, the attributes $a_{i_1}, a_{i_2}, \ldots, a_{i_m}$, where m is the number of the attributes, are not inherited but are defined directly (locally) in the object o_i.

  In the other case of multiple inheritance, if o_{i_1}, o_{i_2}, ..., o_{i_n}, o_j are instances of c_{i_1}, c_{i_2}, ..., c_{i_n}, c_j, respectively, then $(o_{i_1}, a, o_j) \in \mathcal{L}$, $(o_{i_2}, a, o_j) \in \mathcal{L}$, ..., $(o_{i_n}, a, o_j) \in \mathcal{L}$ (Figure 3c) are valid link relationships iff the following is valid:

[1] the value $v_l(t_j)$ can be positioned on the place of the object o_j.

[2] in a case using the value $v_l(t_j)$ in the link relationship, i.e., $(o_i, a, v_l(t_j)) \in \mathcal{L}$, t_j can be positioned on the place of c_j.

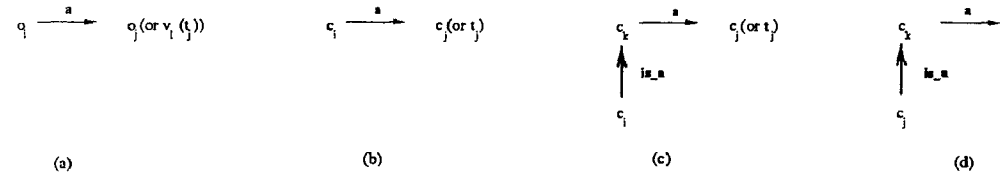

Figure 2: Attribute Link Relationship Cases—Single Inheritance

- *subclass in a place of superclass attributes*, $(c_j, c_{k_1}, c_{k_2}, \ldots, c_{k_n}) \in \mathcal{H}$, $(c_{i_1}, a_1, c_{k_1}) \in \mathcal{P}$, $(c_{i_2}, a_2, c_{k_2}) \in \mathcal{P}, \ldots, (c_{i_n}, a_n, c_{k_n}) \in \mathcal{P}$, and $a_1 \neq a_2 \neq \cdots \neq a_n$, where n is the number of the inherited attributes (Figure 3d). Note that in Figure 3c, the attributes $a_{i_1}, a_{i_2}, \ldots, a_{i_m}$, where m is the number of the attributes which are directly defined in the object o_j, are dropped, and hence, are not shown when the object o_j is subsumed by (placed on the position of) one of its superclasses $c_{k_1}, c_{k_2}, \ldots, c_{k_n}$.

In the first case of multiple inheritance, described above, to resolve the name conflicts among (immediate) superclasses $c_{k_1}, c_{k_2}, \ldots, c_{k_n}$ of the class c_i, respectively, if any two or more of the attributes $a_1, a_2, \ldots, a_n$ have the same name, then the attribute belonging to the superclass $c_{k_l}, l = (1, n)$ which is positioned at the leftmost position (e.g., in Figure 3b—the superclass c_{k_1}) wiil be inherited in the object o_i and in other objects of the class c_i.

- $\mathcal{BL} \subseteq (O \times Mt) \times (O \cup V(T) \times A \times Mt) \times (Mt \times A \times (O \cup V(T)))$ is the behavioral link relationship. If $((o_j, mt_{j_{j1}}), (o_{k_{in}}, a_{k_{j1_{in}}}, mt_{j_{j1}}), (mt_{j_{j1}}, a_{l_{j1_{out}}}, o_{l_{out}})) \in \mathcal{BL}$, then the implementation of the jth operation $op_{j_{j1}}$ of object o_j, the method $mt_{j_{j1}}$ ($j1 = 0, \ldots, n_{mt_j}$, where n_{mt_j} is the number of operations of object o_j), has as input formal parameters the attributes $a_{k_{j1_{in}}}$ ($k = 0, \ldots, n_{j1_{in}}$, where $n_{j1_{in}}$ is the number of the input arguments of the method $mt_{j_{j1}}$) with domain $o_{k_{in}}$, and returns as an output the attribute $a_{l_{j1_{out}}}$ ($l = 0, \ldots n_{j1_{out}}$, where $n_{j1_{out}}$ is the number of the outputs of the method $mt_{j_{j1}}$) with domain $o_{l_{out}}$. An example for this notation is shown in Figure 4. In the case of a single inheritance, where a class inherits the operations of its single superclass, if $o_j, o_{k_{in}}$, and $o_{l_{out}}$ are instances of c_j, c_k, and c_l, respectively, then $((o_j, mt_{j_{j1}}), (o_{k_{in}}, a_{k_{j1_{in}}}, mt_{j_{j1}}), (mt_{j_{j1}}, a_{l_{j1_{out}}}, o_{l_{out}})) \in \mathcal{BL}$—or $((o_j, mt_j), (o_k, a_k, mt_j), (mt_j, a_l, o_l)) \in \mathcal{BL}$, where subindexes are dropped for brevity—is a valid behavioral link relationship, iff one of the following is valid:

  - *new defined operation*, op_j, which is defined in class c_j and is specific to it, namely, $((c_j, op_j), (c_k, a_k, op_j), (op_j, a_l, c_l)) \in \mathcal{O}$ (Figure 5a).
  - *inherited operation*, op_j of class c_j, which is inherited from superclass c_m, i.e., $(c_j, c_m) \in \mathcal{H}$ and $((c_j, op_m), (c_k, a_k, op_m), (op_m, a_l, c_l)) \in \mathcal{O}$, where $op_m, op_j \in Op$. Here, the method name, the names and types of the arguments and outputs, and the contents (implementation codes) of methods mt_j and mt_m are the same (Figure 5b).
  - *not-inherited but overridden operation based on subsumption*, op_j of class c_j, has the same specification (signature) as the one in the superclass c_m of class c_j, i.e., $(c_j, c_m) \in \mathcal{H}$ and $((c_j, op_m), (c_k, a_k, op_m), (op_m, a_l, c_l)) \in \mathcal{O}$, where $op_m, op_j \in Op$. Here, the method name, the name and types of the arguments, and the outputs of methods mt_j and mt_m are the same but the contents, (implementation codes) are different (Figure 5c).
  - *not-inherited but overridden operation with specialization based on subsumption*, op_j of class c_j, which could be used in the place of an operation in superclass c_m of class c_j, i.e., $(c_j, c_m) \in \mathcal{H}$ and $((c_j, op_j), (c_k', a_k, op_j), (op_j, a_l, c_l')) \in \mathcal{O}$. Here, the method name, the name of the arguments, and outputs of methods mt_j and mt_m are the same, but the types of the arguments and outputs, and the contents of the methods are different. The specialization is based on the assumption that if c_j is subsumed into c_m, and $o_j.mt_j(a_k)$ is called, then it is acceptable for the argument a_k to have static type c_k instead of c_k', and for the output a_l to have static type c_l instead of c_l'. However, both of these static types are possible only if c_k is generalized to c_k' and c_l is specialized to c_l'. The generalization is needed for a_k to be used in the context of c_k, that is why, $a_k \in c_k$ must be valid. The specialization is necessary because after the output is produced, a_l will be used in the context of c_l' so that it must inherit all the properties of c_l into its subclass c_l' (Figure 5d).

In the case of multiple inheritance, the validity of the

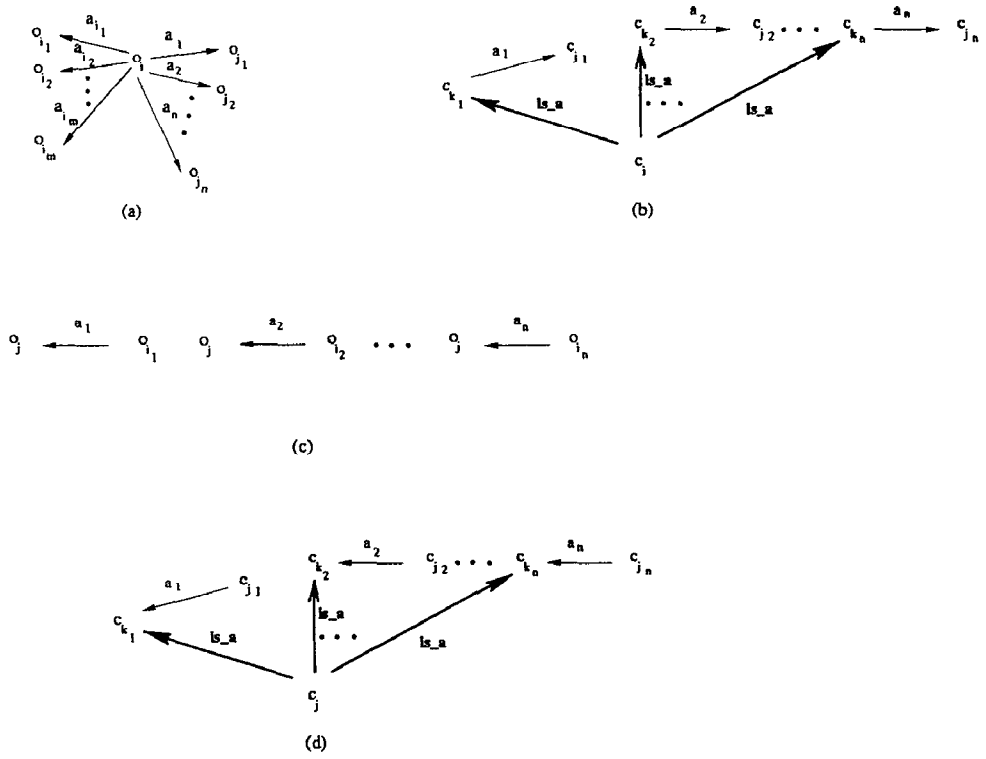

Figure 3: Attribute Link Relationship Cases—Multiple Inheritance

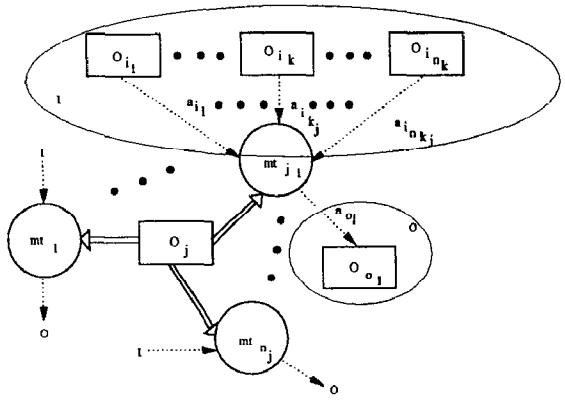

Figure 4: Method Notation

behavioral link relationships is determined in an analogical way, as in the case of link relationships.

The *instance graph*, corresponding to the information system defined above, is a one-to-one correspondent to the conceptual schema graph of the graph-based object-oriented data model. Therefore, the instance graph corresponding to the information system, IS, of the GBOODM model, is represented through a direct labeled graph, $G(IS)$, in which the nodes, N_{IS}, and the edges, E_{IS}, are defined as follows:

- $N_{IS} = O \cup V(T) \cup Mt$ is the set of nodes. If o_j, $v_l(t_j)$, and mt_j are nodes in N_{IS}, they must be in $O, V(T)$, and Mt. Thus, they represent an object, a value, and a method, generated from the corresponding class, built-in type, and operation of the schema Σ, and they are represented by a rectangle, an oval, and a circle in the instance graph, respectively.
- E_{IS} is the set of edges. For each $(o_i, a, o_j) \in \mathcal{L}$, there is an edge e with the label a in E_{IS} from o_i to o_j. For each $((o_j, mt_j), (o_k, a_k, mt_j), (mt_j, a_l, o_l)) \in \mathcal{BL}$, there is an unlabeled method participation edge e in E_{IS} from o_j to mt_j, a labeled method input edge e in E_{IS} with the label a_k from o_k to mt_j, and a labeled method output edge e in E_{IS} with the label a_l from mt_j to o_l.

Based on the conceptual schema and information system of the GBOODM model, introduced in this section, the

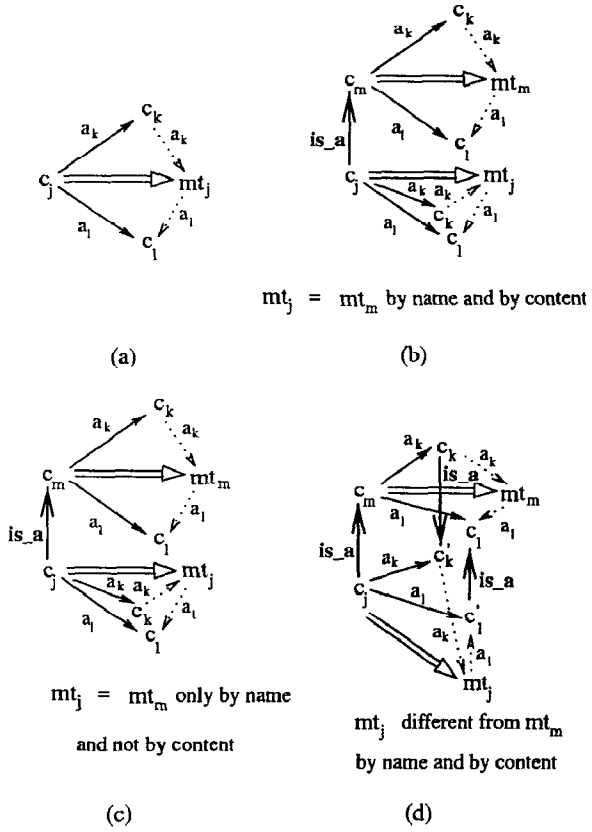

Figure 5: Behavioral Link Relationship Cases—Single Inheritance

schema versioning approach and the perspective version abstraction are introduced in the next section. They are used to handle changing user requirements and the complexity of schema and data representation.

3 Schema Versioning

Schema versioning is based on the notion of a *schema version* $\Sigma\nu$, which is defined as a schema Σ that represents the current result of a successive development, or update, of the original (initial) schema, and reflects the dynamic change of the user requirements towards the data and its schema structure. The schema versioning approach used in the GBOODM model is based on specification of a complete initial schema version and subsequent derivations of new schema versions out of it and out of its descendants. During the process of derivation of a new schema version, the consistency of the schema and the propagation of the corresponding instance base are automatically preserved by using schema derivation graphs, instance conversion functions, and instance propagation flags that will be introduced later on in this section.

The derivation of each new, *target schema version* is realized by applying a set of schema update primitives to the schema which is used as a *source schema version*. The derived schema versions are ordered and stored in a direct acyclic graph, called *schema derivation DAG*. The graph contains information about the relationships among the existing schema versions.

The process of building the schema derivation DAG includes the following. The initial schema of a user application serves as the root of the derivation DAG of this schema. Depending on the user requirements, a source schema version $\Sigma\nu$ is selected. Any of the schema versions in the DAG can be used as a source schema version. Then, the schema update primitives are applied to the selected source schema version to obtain the target schema version $\Sigma\nu'$. After that, specifications for each class in the source schema version are provided. They direct the propagation behavior of the objects of that class which exist before the derivation. They also direct the behavior of the class objects to be created after the derivation finishes. When the schema derivation completes, what is called *derivation time* of $\Sigma\nu'$, the target schema $\Sigma\nu'$ is included in the schema derivation DAG as a child of the source schema version $\Sigma\nu$. It can be further used for other schema derivations.

In order to define each schema update primitive, the part of the source schema version to which it is specified (attached) must be selected. For this reason, *schema pattern*, Π, is defined. The pattern is part of the source schema version, $\Sigma\nu$, over which the schema update primitives are applied to derive the target schema version, $\Sigma\nu'$. The pattern is selected as a set of paths over the source schema version. Each of the paths is defined as the regular expression $((\mathcal{H})^*\mathcal{P}|(\mathcal{H})^*\mathcal{O}\wp\mathcal{S}(\mathcal{IN})^*\mathcal{OUT})^*$. This means that a path is a sequence of edges such that an inheritance edge ($\mathcal{H}$) can only be followed by an attribute edge ($\mathcal{P}$), another inheritance edge, or by an operation participating edge ($\mathcal{O}\wp\mathcal{S}$), operation input edge ($\mathcal{IN}$), or an operation output edge ($\mathcal{OUT}$), and the path cannot end with an inheritance edge. The pattern Π is represented as the graph $G(\Pi) = (N_\Pi, E_\Pi)$, where $N_\Pi \subseteq N_{\Sigma\nu}$, $E_\Pi \subseteq E_{\Sigma\nu}$, and $G(\Sigma\nu) = (N_{\Sigma\nu}, E_{\Sigma\nu})$ is the graph representation of the source schema version $\Sigma\nu$.

The initial schema version, $\Sigma\nu$, and its corresponding set of instances, called *instance extent version* $\mathcal{I}\nu$, both represent the initial user perspective on the database schema and content. On the one side, this initial representation can be gradually customized by the users' changing requirements, but on the other side, it can become quite complex in many applications, including multimedia ones. Hence, both of these requirements need to be satisfied using means to handle them. In the GBOODM model, the means of schema evolution using the schema versioning approach, handles the frequently changed user requirements, while the mechanism of perspective version handles the reduction of the representation complexity in addition to the changing user requirements.

3.1 Perspective Version

The *perspective version* $P(\Sigma\nu, \mathcal{I}\nu)$ consists of a schema version $\Sigma\nu$ derived directly, or indirectly, from the original schema through schema versioning, and an instance extent version $\mathcal{I}\nu$ corresponding to it. As an example, let us assume that the schema version $\Sigma\nu'$ shown in Figure 6a is the target schema version which is the result of several schema derivations starting at the original schema (schema version) $\Sigma\nu$ shown in Figure 1. The corresponding instance extent version $\mathcal{I}\nu'$ could then look like the one shown in

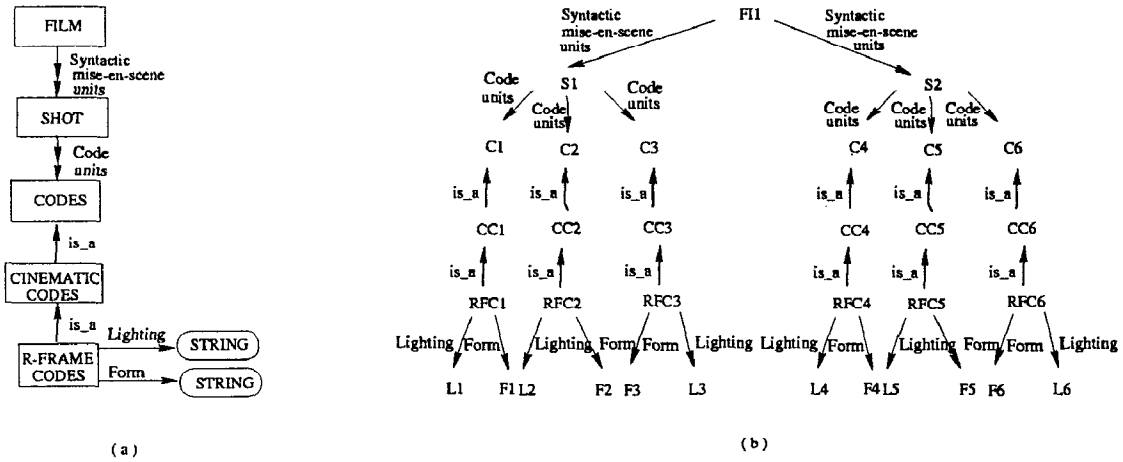

Figure 6: Perspective Version $P'(\Sigma\nu', \mathcal{I}\nu')$

Figure 6b. Both together build the perspective version $P'(\Sigma\nu', \mathcal{I}\nu')$, which focuses on those representative frame (r-frame) codes, or elements, of a film shot (a sequence of video frames shot with one uptake of a camera) that reflect the form and lighting with which film shots were built. Thus, the complexity of the information initial representation in the original perspective schema version, which models the entire filmic syntax, has been reduced to the representation of only those elements (form and lighting of film shots) that the user is interested in when he uses this perspective version.

Once the user scope has been reduced to the information contained in the perspective version, then by using the mechanism of *filtering*, the attention of the user could be further restricted to the information contained in a specific part (which is a subset of instances) of the perspective instance extent version. The filtering mechanism allows visual queries to be posed.

A filter contains a set of conditions defined over the perspective schema version $\Sigma\nu$. The conditions serve for selection of those instances of the corresponding perspective instance extent version $\mathcal{I}\nu$ whose values match these selection conditions. The selection conditions are built over the attributes of selected classes in the perspective schema version $\Sigma\nu$. They are built from values (constants or operands) of the domains of the built-in types in T. These values, in the case of operands, are combined, or compared, using logical operators of conjunction and negation, $\wedge$ and $\neg$, respectively, or other comparison operators depending on the domain $t \in T$ of the operands.

Over the conceptual graph of the schema version $\Sigma\nu$, the filters are expressed, first, by coloring (shadowing) the class nodes and their belonging attributes over which the filter is formulated, and then, by entering the values and the operators of the selection conditions. Formulating a filter over the graph of the perspective schema version $\Sigma\nu$, represents the formulating of a *visual query* against the perspective instance extent version $\mathcal{I}\nu$. If there are several selection conditions built over different attributes of the same class or different classes of the schema version, all of the conditions must be satisfied (i.e., matched successfully against the values of the corresponding attributes of the instances in the instance extent version, in order for these instances and the other instances reachable starting from them to be retrieved).

As an example, a filter over the perspective schema version, shown in Figure 6a, is defined in Figure 7a by requesting the instances with values $F4$ and $L4$ of the attributes *Form* and *Lighting* of the class *r-frame codes* and the other reachable instances to be retrieved. Assuming that the values $F4$ and $L4$ are actually the *open form* and the *backlighting*, respectively, this visual query is the equivalent of the following query: "Find the shots from all films of the film database that possess representative frames with both an

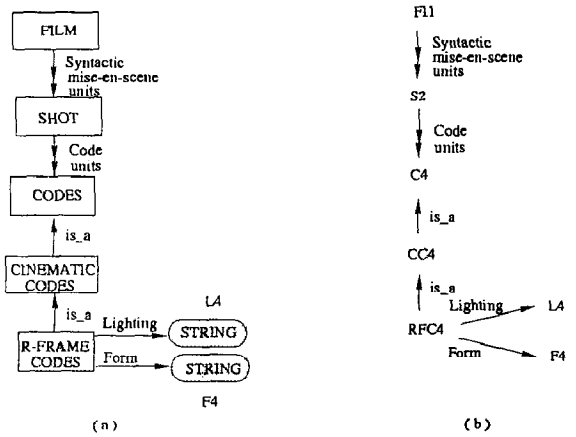

Figure 7: Result of the Query Defined on Perspective Version $P'(\Sigma\nu', \mathcal{I}\nu')$

open form and a backlighting and visualize these frames." For visualization of these frames, values of the *Visualization* attribute of the class *r-frame codes* are assumed to be present in the filter and stored in the video database so that they can be retrieved as a result of the evaluation of this query; however, they are not shown in the figures. The result of this visual query, (i.e., the matching instances from the instance extent version $\mathcal{I}\nu'$ in Figure 6b) is shown in Figure 7b.

In this subsection, the mechanism of perspective versioning and its correlating mechanism of filtering were exposed. They handle the continuously changing user requirements, reflecting the change of the user interests, or user perspective, and the reduction of the complexity of perspective version representations. They do not handle, however, the transformation from one perspective version to another, which is done to reflect the change of the user perspective over the information contained in the database. In the GBOODM model, this is done using the mechanism of schema evolution based on the schema versioning approach, and it is discussed in the next subsection.

3.2 Framework of Schema Versioning

The schema update primitives of the GBOODM model used to derive a target schema version $\Sigma\nu'$ from a source schema version $\Sigma\nu$ are the basis of the schema versioning approach of the GBOODM model. Their semantics are based on a framework, consisting of a set of properties (invariants) of the class inheritance hierarchy and a set of rules. The rules serve as a guidance for the selection of the most proper way of enforcing an invariant after applying a SUP when there are more ways than one to do so. The set of invariants and the set of rules used in the GBOODM model are similar to those in the existing object-oriented systems, for example those in Banerjee et al. [1]. The invariants must hold for every schema version before and after its derivation time, i.e., before and after the application of the SUPs. Therefore, they provide consistency of each schema version. The invariants include the class lattice invariant, distinct name invariant, distinct identity invariant, full inheritance invariant, and domain compatibility invariant [1]. The definitions of the schema update primitives of the GBOODM model are now presented together with their semantics to illustrate their employment in the GBOODM model.

3.3 Schema Update Primitives

In the definitions of the schema update primitives for a given SUP, the pattern Π is called the *source schema pattern* of this SUP, and it has the graph representation $G(N_\Pi, E_\Pi)$. Its corresponding instance base $\mathcal{I}\Pi$ is called the *source instance base pattern*, and it has the graph representation $G(N_{\mathcal{I}\Pi}, E_{\mathcal{I}\Pi})$.

In the GBOODM model, typically, all SUPs are executed over the graph of the target schema pattern Π'. The (graphical) parts (items) in Π' which correspond to these SUPs and which are not present in the source schema pattern Π are marked in bold in the case of an addition of a part (a class node, an edge, etc.). At a given time, more than one SUP can be attached to a given source schema pattern Π, and more than one pattern Π over the source schema version $\Sigma\nu$ can be defined. After the execution of the set of all SUPs defined over the source schema version $\Sigma\nu$, a new derived schema version $\Sigma\nu'$ is generated.

A guiding principle of the semantics of all SUPs which deal with addition (deletion) of a part is that when a part is added (deleted), all the other parts which depend on, or belong to, this part are added (deleted) also. As an example, the definition of a SUP for addition of a part is considered next.

Class Node Addition (CNA)

The class node addition operation,

$$CNA(\Sigma\nu, \Pi, \mathcal{I}\Pi, c_{\Pi'}, \{(c_{\Pi'}, a_{\Pi'_1}, c_{\Pi_1}), \ldots,$$

$$(c_{\Pi'}, a_{\Pi'_n}, c_{\Pi_n})\}, \mathcal{I}\nu\_PF, \mathcal{I}\Pi\_CF)$$

produces the new, target schema pattern Π' and its corresponding new, target instance base pattern $\mathcal{I}\Pi'$ over the new derived, target schema version $\Sigma\nu'$. The new instance extent version $\mathcal{I}\nu'$, corresponding to the new derived schema version $\Sigma\nu'$, is also generated, using the propagation flags, $\mathcal{I}\nu\_PF$, and conversion functions, $\mathcal{I}\Pi\_CF$, and thus, the new perspective version $P'(\Sigma\nu', \mathcal{I}\nu')$ is derived. The flags $\mathcal{I}\nu\_PF$, called *instance propagation flags*, are defined for each class in the source schema version $\Sigma\nu$.

The functions $\mathcal{I}\Pi\_CF$, called *instance conversion functions*, are defined for each class in the source schema pattern Π.

The class $c_{\Pi'}$ is a class which participates in the target schema pattern Π', while the set of edges $\{(c_{\Pi'}, a_{\Pi'_1}, c_{\Pi_1}), \ldots, (c_{\Pi'}, a_{\Pi'_n}, c_{\Pi_n})\}$ are attribute edges in the pattern Π'.

The target schema pattern $\Pi' = (C_{\Pi'}, A_{\Pi'}, Op_{\Pi'})$ is obtained from the pattern $\Pi = (C_\Pi, A_\Pi, Op_\Pi)$ by adding to C_Π the class $c_{\Pi'}$ and by adding to A_Π the attributes $a_{\Pi'_1}, \ldots, a_{\Pi'_n}$. The graph $G(N_{\Pi'}, E_{\Pi'})$ of the target pattern Π' is obtained from the graph $G(N_\Pi, E_\Pi)$ of the source pattern Π in the following way:

- the nodes—$N_{\Pi'} = N_\Pi \cup node\_of c_{\Pi'}$
- the edges—$E_{\Pi'} = E_\Pi \cup (c_{\Pi'}, a_{\Pi'_1}, c_{\Pi_1}) \cup \cdots \cup (c_{\Pi'}, a_{\Pi'_n}, c_{\Pi_n})$

The new added attributes must have different names than the names of the rest of the attributes in pattern Π, i.e., the distinct name invariant must be satisfied.

The target schema version $\Sigma\nu'$ is obtained from the source schema version $\Sigma\nu$ as $\Sigma\nu' = (\Sigma\nu \backslash \Pi) \cup \Pi'$, where $\Sigma\nu \in \Sigma\nu'$.

The graph $G(N_{\Sigma\nu'}, E_{\Sigma\nu'})$ of the version $\Sigma\nu'$ is obtained from the graph $G(N_{\Sigma\nu}, E_{\Sigma\nu})$ of the version $\Sigma\nu$ in the following way:

- the nodes—$N_{\Sigma\nu'} = (N_{\Sigma\nu}\ N_\Pi) \cup N_{\Pi'}$
- the edges—$N_{\Sigma\nu'} = (E_{\Sigma\nu}\ E_\Pi) \cup E_{\Pi'}$.

The target instance extent version $\mathcal{I}\nu'$ is obtained using the propagation flags, $\mathcal{I}\nu\_PF$, and conversion functions, $\mathcal{I}\Pi\_CF$. The concepts of propagation flags and conversion functions in the GBOODM model are similar to those used in the system presented in Lautemann and Wöhrle [3]. Due to the space limitation, their use in the GBOODM model and the rest of the SUPs will be discussed in a future paper.

4 Conclusion and Future Work

In the present paper, the characteristics of the novel graph-based object-oriented data model were presented. This model represents the structural and behavioral aspects of data that form multimedia information systems. It also provides for handling the continuously changing user requirements and the complexity of the schema and data representation in multimedia information systems using the schema versioning approach and perspective version abstraction.

The schema update primitives used in the model can be further exploited to provide a full-fledged schema evolution in a graphical manner. These primitives are low level primitives because they are closely tight to their data model (GBOODM). More abstract primitives, higher level primitives, can be investigated for integration with the GBOODM model. They will free the programmer from the need of handling the design details and steps, as required by the low level primitives, and will allow declarative advanced operations such as generalization or merging of schemas during the design of MMISs.

Today, the Web is a rich source of semistructured data [2], where there is no clear separation of schema and data, and where the information is naturally modeled by graphs. A significant part of the semistructured data in the Web is multimedia oriented. Handling of this type of data requires novel approaches. That is why, the graph nature of information modeling and the facilities for reduction of the representation complexity of the multimedia information, both provided by the GBOODM model, can be further investigated for their application in semistructured databases. This will allow a modified version of the GBOODM model to handle the semistructured information flowing in the Web.

References

[1] J. Banerjee, H.-T. Chou, J.Garza, W. Kim, D. Woelk, and N. Ballou. Data Model Issues for Object-Oriented Applications. *ACM Transactions on Office Information Systems*, 5(1):3–26, January 1987.

[2] P. Buneman. Semistructured Data. In *Proceedings of ACM Symposium on Principles of Database Systems*, Tucson, Arizona, 1997.

[3] S. Lautemann and C. Wöhrle. The Coast Project Design and Implementation. In F. Bry and R. Ramakrishnan, editors, *Proceedings of Deductive and Object-Oriented Databases (DOOD 97)*, volume 1341 of *Lecture Notes in Computer Science*, pages 229–246, Montreux, Switzerland, December 1997. Springer-Verlag.

[4] M. Levine and G. Loizou. A Graph-Based Data Model and its Ramifications. In *IEEE Transactions on Knowledge and Data Engineering*, 1995.

[5] D. Lucarella and A. Zanzi. A Visual Retrieval Environment for Hypermedia Information Systems. *ACM Transactions on Information Systems*, 14(1):3–29, January 1996.

[6] I. Radev, N. Pissinou, and K. Makki. Film Video Modeling. In *Proceedings of IEEE Workshop on Knowledge and Data Engineering Exchange, KDEX 99*, Chicago, Illinois, November 1999.