A Metadata Approach to Multimedia Database Federations

Damir Bećarević and Mark Roantree
Interoperble Systems Group, Dublin City University, Dublin 9, IRELAND.
{damirb,mark}@computing.dcu.ie

Abstract

Recent research in federated database systems has advanced in the direction of federations of multimedia databases. However, in each project studied, there has been little emphasis placed on the subject of meta-models. Conversely, standard database systems (relational, object-relational and object-oriented) define metadata models that lack multimedia and federation features. In this paper, a specification of a metadata architecture for a multimedia database federation is presented. The architecture is based on a new object-oriented metamodel that has support for multimedia types and federated metadata. By using this metamodel, administrators and users of a multimedia federation can determine precisely how data is structured and the types of operations that are possible.

Keywords: Federated Databases; Metadata; Multimedia.

1 Introduction

Many applications require access to databases of unknown structure where the initial objective is to determine the database entities and their relationships prior to access or modification of data. In the area of federated databases, this requirement is stronger as a global administrator or engineer must take multiple heterogeneous software systems, determine similarities and differences across systems, and combine them to form one or more global schemas. This type of genericity in database access is commonly referred to as reflection, which denotes the possibility of dynamic construction and invocation of requests [12]. A key support feature in this task is the presence of a schema repository where the database structure is described, and a metamodel which provides a set of legal relationships and actions for entities in the database.

In the EGTV (Efficient Global Transactions for Video media) project [19], the goal is to integrate a medium to large number of multimedia database systems through a common metamodel interface. The contribution of this research is in the specification of a metamodel for multimedia federations, of which an implementation has been mapped to both the object-oriented database standard (ODMG) and Object Relational (O-R) metadata models. Furthermore, there is an emphasis on the metadata aspects of integrating large multimedia systems. We argue that by specifying full details of the metamodel (or schema of the database repository), this enhances the integration process as it provides clearly how to interrogate the schema.

This paper is structured as follows: The remainder of this section provides a background and motivates the problem. Related research in distributed multimedia systems are presented in §2, while §3 presents the operational architecture of our system. §4 describes the metadata model specification, and §5 discusses mappings to the object-oriented and object-relational metamodels. Implementation is described in §6, while conclusions are provided in §7.

1.1 Background and Motivation

Improvements in processing power and storage capabilities of modern computers have lead to the development of new multimedia applications. One such application is the Fischlár [13] video indexing system. This system provides content indexing of television programs recorded in form of MPEG digital videos. The goal is to make browsing of large video contents easier by automatically detecting scene and shot boundaries in the video file. This information is later used for random navigation and browsing of the recorded material.
The main obstacle to expansion of the system is the inability of its video storage subsystem to cope with increasing amounts of recorded data. At present, all video files can be stored only at a central file-based repository which does not provide any database capabilities and has limited storage capacity. This leads to an inability to archive video data, as old videos must be continuously deleted to find space for new recordings. The present storage system also lacks the capability of organising video data in complex multimedia schemas suitable for advanced queries.

The EGTV (Efficient Global Transactions for Video Media) project is aimed at providing an efficient database storage system for multimedia in a distributed architecture. It cannot be assumed that a single centralised data storage will suffice, but instead, a more powerful database architecture must exist in which multiple digital hubs may share multimedia. One solution, proposed in the EGTV project, is to use standard object-oriented and object-relational databases for federated storage of large multimedia data objects. Databases in the federation physically store multimedia or act as object wrappers for proprietary data stores. The advantages of using a federated architecture are the ability to integrate large amounts of multimedia distributed across multiple databases, and define global schemas by creating object views of locally stored multimedia data. Multiple decentralised database sites increase the overall storage capacity of the system and eliminate the need for continuous deleting of recorded videos. The federated architecture enables definition of personal multimedia databases which can be later integrated to global schema. The advantage of the federation is that many autonomous users can browse a large number of multimedia repositories through single federated schema.

The metamodel plays a key role in the construction of this system since it describes the database model of each multimedia database and defines local view schemas which are then integrated to form one or more global (federated) schemas. A metamodel schema is usually stored within the database in a special segment called the schema repository and includes a catalogue of system data types and their properties, user defined entities such as tables or classes, relationships and database behaviour. Additional metadata may include data distribution information, database users and security rules. This is structural metadata and it is different from the MPEG-7 [15] metadata which describes multimedia content. Object-relational databases define a schema repository in the form of extensions to existing relational metamodel. Although it can represent object types and object tables, the object-relational schema repository itself is implemented as a set of relational tables. The main standard for object-oriented databases is defined by the Object Data Management Group (ODMG) [6]. The ODMG specifies the schema repository as a set of abstract interfaces, where each interface defines one component of the object-oriented model. The metamodel presented in this specification is complex, with a large number of interfaces and association links between them. Interface definitions are cumbersome, and the same information is repeated at different places in the metamodel. So far, none of the commercial ODMG database vendors have implemented this schema repository.

Commercially available database metamodels are mutually incompatible and interoperability between them cannot be easily achieved. Furthermore, none support multimedia databases and object view schemas. These are the main problems which motivate our effort to define a new metamodel specification for multimedia database federations.

2 Related Research

In this section we present an overview of research in the area of distributed multimedia systems and metamodel architectures for database systems. Issues addressed in these projects include multimedia data and metadata representation, query languages for multimedia and federated multimedia architectures.

2.1 The Fischlár Digital Video Recording and Browsing System

The Fischlár Digital Video Recording and Browsing System [13] is a multimedia system for digital video indexing and browsing. The system enables clients to record television programs and watch previously recorded videos through a web interface. Selected TV programs are recorded in the MPEG-1 format and stored in the regular file system. Recorded videos are then processed by video indexing software which performs shot and scene boundary detection and generates web pages with thumbnail images representing the beginning of each scene within the video. Clients can select any of these images to begin video playback start-
ing from that particular scene. Real-time video streams are then broadcast over the TCP/IP network from the video-streaming server to clients. A streaming server (Oracle Video Server) stores MPEG-1 video files in a specialised file system optimised for fast retrieval, while scene and shot indexing information are fully contained in the web interface.

**Summary.** The video indexing capability represents the most important feature of the Fischlár system. Automated shot and scene detection processes provide fast indexing and enable easy browsing of large video files. The main problem with this system is the lack of support for data distribution since the present storage system can store video files in a single centralised repository only. Since MPEG-1 files consume large amounts of space, problems like storage capacity, fast retrieval and real-time streaming can be expected with the growth of the system. Data distribution using multiple inexpensive databases provide one viable solution to this problem.

**2.2 The Garlic Project**

The Garlic Project [5] defines an architecture for a distributed database system that can store and manipulate heterogeneous multimedia data. It provides a global schema for multimedia data originating from different data stores, where a data store can be any database or specialised multimedia storage system. A global schema is a union of local schemas transformed to the Garlic data model. The Garlic data model is based on the ODMG model but is extended with object view support. Views can extend, simplify or reshape class properties and methods, but each view can be formed from only a single base class. Objects originating from different data stores can be combined using complex objects. Complex objects are stored in the special Complex Object Repository and model relationships that exist between multimedia objects. The query language for Garlic is based on SQL and extended with object-oriented features like references, collections and operations. These extensions include predicates and operations for context querying of multimedia objects. A global query is decomposed to set of smaller queries that are executed on the wrapper databases.

**Summary.** The global schema and object based data model for multimedia representation are the most important features of the Garlic architecture. Different multimedia data stores can be joined to Garlic system by using object wrappers that provide translation to Garlic data model. The object-oriented query language has advanced querying capabilities, but no transaction support is provided. Queries are read only and data modification is not possible. Limited view capabilities are an other disadvantage of the architecture. The main issue with Garlic is its inability to define relationships between classes in the global schema. Relationships are substituted by the concept of complex objects that must be stored in the special repository. A metadata model was not included in any of the literature covered.

**2.3 The News-On-Demand System**

The News-On-Demand System [18, 25] is as a multimedia project developed at the University of Alberta in Canada. Each news document consists of textual and multimedia elements with spatial and temporal relationships. The spatial relationships between multimedia elements are represented in SGML, while the Hy/Time standard was used for temporal relationships. From a database aspect, the system provides an object-oriented representation for multimedia elements. Non-continuous media (text and still images) are stored in the ObjectStore database, while continuous media (audio and video) are stored in a special media file server. The architecture does not provide heterogeneity, since all objects must be stored in these two data stores. A query language for multimedia was also developed as part of the project. The language is based on ODMG OQL, and extended with multimedia features (MOQL). This includes functions and predicates for spatial and temporal querying of multimedia data. However, the language does not support updates or transactions. A detailed specification of the MOQL language can be found in [14].

**Summary.** The advantages of this system are advanced data model and multimedia extensions for the OQL query language. The MOQL language extensions enable advanced querying by spatial, temporal and presentation relationships between multimedia objects. The disadvantage is that each user-defined multimedia class must derive from the multimedia class hierarchy defined in the data model. The MOQL language is query only, and updates are not supported. No metadata model was specified in any literature covered.
2.4 The Federated Multimedia Database System

The Federated Multimedia Database System [4] comprises textual and multimedia data into a federation which maintains a single global schema. Local data stores connect to the system through wrappers that provide standardised interfaces to local data. A global federated schema is constructed in two steps. First, two intermediary schemas are constructed containing media and structured (non-mediain) data schema. A structured data schema is a union of all schemas exported from databases that store regular non-mediain data, while the media schema represents a union of all schemas exported from the local media stores. The media schema does not physically store multimedia objects, but contains proxies that map to real multimedia data. In the second step of global schema construction, multimedia and structured data schemas are integrated in the global schema and stored in a special repository called internal database. The integration is accomplished by establishing relationships between multimedia and structured classes. New complex multimedia classes can be constructed in the global schema. These classes integrate one or more single multimedia classes and include spatial and temporal relationships.

Summary. The advantage of this system is that it constructs a database federation that integrates multimedia data and regular data. Multimedia classes in the data model are defined as a class hierarchy and complex relationships can be defined between multimedia classes. The disadvantages of this system include the lack of a metamodel and query language for multimedia. Construction of the global schema is complex because the multimedia schema and structured data schema are constructed separately and then integrated to form the global schema.

3 The EGTV System Architecture

The architecture presented in this section facilitates the construction of a global schema for integrating different multimedia data sources into a database federation. It is based on the standard architecture for federated database systems [23] with some modifications required for multimedia data handling. The architecture can support large multimedia libraries from many inexpensive general purpose databases. The emphasis is on metadata which has significant importance in the construction of the federated system, since it is required for the construction of a global schema and for generic querying.

The architecture, illustrated in Figure 1 consists of five layers, where each layer is defined in the form of a database schema. Schemas are constructed and manipulated by processors that are located between layers. Our architecture differs from the generic five layer architecture [23], in several aspects. First, multimedia data stores at the Database Layer are restricted to the ODMG object-oriented and object-relational databases. Second, the canonical schema is defined in a form of EGTV metamodel representation and it uses two different processors (Query Processor and Transformation Processor) to interact with the Database Layer. Objects instantiated from the canonical schema are represented in the platform independent EGTV model representation [11]. Finally, the External Layer provides clients with both CORBA and XML interfaces for federated schema access.
• **Database Layer.**
In the federation, all data is stored in either ODMG or object-relational databases at this layer. Databases are used for physical storage of data and multimedia objects, but can also provide an object wrappers for proprietary multimedia data stores.

• **Canonical Layer.**
The Canonical Layer contains both data and metadata in a common representation. The canonical schema is represented in the EGTV metamodel format [20]. This layer is the entry point for database schema definition and for local data access. From the user’s perspective, the Database Layer is completely encapsulated and accessed through a single common interface.

• **View Layer.**
The View Schema is that subset of the Canonical Schema to be shared within other databases. It is expressed in a form of object views. For further information on the object view model used in the EGTV multimedia federation please refer to [21].

• **Federated Layer.**
The Federated Schema is an integration of multiple view schemas. It stores federated metadata in the EGTV model format and represents an access point for global queries.

• **External Layer.**
The role of the External Layer is to transform Federated Schema or any of its subsets to representation suitable for client access. The External Layer in our architecture provides XML and CORBA interface access to the database federation. The XML Interface and CORBA Interface components are described later in this section.

• **Transformation Processor.**
The Transformation Processor translates the data and metadata model of a local database to the canonical representation. The transformation rules for mapping the EGTV metamodel to ODMG and object-relational formats are discussed in §5.

• **Query Processor.**
The Query Processor transforms queries submitted against the Canonical Layer schema to the representation of each local database. Query results encoded in the EGTV model format are made available to the Canonical Layer. This processor is also responsible for propagating updates on the Canonical Layer schema back to the local database.

• **Filtering Processor.**
The Filtering Processor creates view subschemas by filtering and restructuring metadata definitions stored at the Canonical Layer. The ODLv language [21] is used for defining the view subschemas.

• **Integration Processor.**
The role of the Integration Processor is to create federation by joining multiple View Schema definitions to a Federated Schema. This also uses the ODLv language for defining federated view subschemas.

• **CORBA Interface.**
Clients of the federated system use CORBA interface for compile-time programming language access to the Federated Schema. The interface creates CORBA proxies for objects in federated schema and enables clients to access them from within the programming environment. The discussion on CORBA interface is out of the scope of this paper, as it is part of the separate research work [10].

• **XML Interface.**
The XML Interface provides generic query access to the Federated Schema. The interface processes queries received from the clients, and returns data and metadata results in the XML representation. Due to inefficiency of XML in representing binary data, multimedia is encoded and transported in the pure binary format. The query language for multimedia is part of our current research.

4 **The EGTV Metamodel Specification**

This section describes the object-oriented metamodel designed for multimedia databases, and which has a special emphasis on integration of multiple databases. In this context, the metamodel defines an object-oriented meta-schema for representing textual and multimedia metadata for databases at both the canonical and federated layers. For space reasons, we limit the
discussion of each subset of the metamodel to a brief overview: please refer to [1] for a more detailed description, including a breakdown of the ODMG metaclasses which have been eliminated or amended, and those which are new metamodel specification.

The EGTV metamodel is based on the ODMG metamodel specification [6] and the C++ implementation defined in [9]. Our metamodel eliminates ambiguities and redundancies present in both specifications by clearly defining the metamodel structure, and significantly reducing the overall complexity. Some modifications required for the representation of multimedia data types and behaviour were also introduced. Our metamodel does not incorporate metadata access interface (as the ODMG metamodel does) because this has been shown to limit generic query capabilities [8]. All metaclasses in our metamodel have the sys prefix.

The specification defines the abstract, platform independent representation of the metamodel structure. The actual implementation of metaclasses is not discussed here as it depends on the type of database where the metamodel schema repository implementation is located. The metamodel mappings developed for the object-relational and ODMG object-oriented databases specify platform specific implementation details, and they are presented later in this paper.

4.1 Defining Name Scopes

![Figure 2: Metadata Definition of Name Scopes.](image)

Each database entity has a name that must be unique within the scope to which it belongs. For example, attribute names are uniquely defined within the containing class, while class names are unique in the database schema. Name scopes and containment relationships in the metamodel are closely related because each metaclass provides a scope for all its contained elements. Our metamodel defines a single containment/scope relationship between metaclasses as illustrated in Figure 2.

The sys_MetaObject is an abstraction for all elements in the metamodel and defines common attributes such as name, metaclass type (metaType) and comment. These three attributes provide name, type and user defined comment properties for all elements stored in the metamodel. Metaclasses that are not capable of containing other metamodel elements, like sys_Property, sys_Parameter and, sys_Inheritance derive directly from sys_MetaObject, while all container metaclasses derive from sys_ScopedObject. The sys_ScopedObject defines the containment relationship (contains) to sys_MetaObject, so that each instance of sys_ScopedObject can contain and define name scope for many sys_MetaObject instances. For example, an instance of sys_ScopedObject can contain attributes, relationships and operations which are uniquely identified within the scope of that class.

Since the sys_ScopedObject metaclass also derives from the sys_MetaObject metaclass, each container metaclass can recursively contain another container class. Metaclasses sys_Class, sys_Schema and sys_Operation derive from the sys_ScopedObject, as they provide naming scope for elements contained within them. The top level scope is the database schema itself (sys_Schema), and it contains the classes (sys_Class) defined by users.

4.2 Defining Types

Type metaclasses derived from sys_ScopedObject provide a description for types defined in the database. All data types are represented in a metadata class hierarchy as illustrated in Figure 3. The sys_Type metaclass is an abstraction for all types in the database, and the more specific metaclasses which derive from it. Our metamodel is enhanced by permitting operations for the sys_Type metaclass, whereas the ODMG metamodel permits only user defined classes to have operations. Moving operation support to the level of the sys_Type base class enables the definition of operations not only for classes, but also for other data types (e.g. multimedia and collection types). The importance of this feature arises from the fact that the internal structure of
complex data types is fully encapsulated and the only interface is provided through operations. For example, a jpegImage multimedia data type can have operations for query by pattern, resizing and rotating of an image it contains. These operations are registered in the metamodel and publicly available, while the implementation and storage details of the jpegImage data type are hidden from the user.

The system and user defined types are represented as a specialisation of the sys_Type metaclass. System types are used as attributes of classes, or parameters of operations, and they cannot be instantiated to persistent self contained database objects. System types can achieve persistence only as attributes of user defined classes. All system types can be classified as primitive and collection types.

- **Primitive types**: Represented in the sys_PrimitiveType metaclass. The full set of allowable types are Integer, Float, String, Date and Boolean. Special data types for multimedia storage are incorporated into this metamodel and they are represented as instances of the sys_MediaType metaclass. The metamodel defines the sys_MediaType metaclass as a specialisation of the sys_PrimitiveType. The mediaKind property identifies multimedia type (audio, video, text or image), while encodingFormat, formatVersion and compression provide information on media encoding characteristics.

- **Collection types**: Collections store multiple objects of one system type and are represented by the sys_CollectionType metaclass. Supported collection types include Bag, List and Dictionary as specified in the ODMG standard.

The sys_Class metaclass represents user defined type and corresponds to d_Class in ODMG metamodel. Classes can contain attributes, relationships and operations for which they provide scope. All classes defined in the metamodel are classified in two categories: base classes and virtual classes. Base classes can be instantiated to persistent database objects that store data, while virtual classes are components of object views and they are constructed from the base classes. Virtual classes and object views are explained in the §4.7. The isAbstract property is applicable only to base classes and it specifies if class is defined as an abstract one.

### 4.3 Defining Properties

The sys_Property metaclass is an abstraction for all property types that can be specialised as attributes or relationships and it is derived directly from the sys_MetaObject. Members of a class are specified in the sys_Attribute metaclass, where each attribute can be optionally defined as static or constant using isStatic and isConstant properties. Attributes can be of a system or a class type, where each attribute has only one type, while one type can be used by many attributes. This is represented with the attribute_etype relationship between sys_Attribute and sys_Type metaclasses where the sys_Type is a superclass for all types in the metamodel.

The sys_Relationship metaclass defines bidirectional relationships between two classes where a cardinality of one or many is specified for each side of the
relationship. The traversal property of the `sys_Relationship` returns the other side of the bidirectional relationship. Each relationship with cardinality greater than one can have ordered values (`isOrdered` property) or it can be defined as unique (`isUnique` property).

### 4.4 Defining Inheritance

![Figure 5: Metadata Definition of Inheritance.](image)

Inheritance relationships are defined for classes only. Inheritance in the metamodel is represented by the `sys_Inheritance` metaclass which inherits from the `sys_MetaObject`. The instance of `sys_Inheritance` has an `inherits_to` and a `derives_from` relationship with two instances of `sys_Class`. Each class has a list of inherited classes and a list of derived classes. The `positionNumber` parameter indicates the order of base classes in multiple inheritance definitions.

### 4.5 Defining Operations

![Figure 6: Metadata Definition of Operations.](image)

Operations can be defined for both system types and classes. Operations specified for system types are part of the type definition and cannot be modified by the user, while operations on classes are user defined. The `sys_Operation` metaclass is an abstraction for all operations defined in the database and it is derived from the `sys_ScopedObject` metaclass. This metaclass corresponds to the `d_Operation` class in the ODMG metamodel with the difference that in our metamodel, `sys_Operation` is generalisation for two types of operations: methods and operators defined by `sys_Method` and `sys_Operator` metaclasses respectively. Operators are not supported in the ODMG metamodel and they are new to our metamodel. An operator can be unary or binary as defined in `operatorKind` property, while `methodKind` property of the `sys_Method` indicates if method is prefixed as static or virtual.

Each operation can have a list of parameters and a return value. Parameters are specified by the `sys_Parameter` metaclass and they can be of any system type. Classes cannot be passed by value as a parameters or as a return values of operations, and class references are passed instead. The `positionNumber` attribute of the `sys_Parameter` metaclass indicates the relative position number of the parameter within the parameter list. Operations can be public, private or protected as specified in the `accessKind` property. Each operation can be defined as constant (`isConstant` property), indicating that the internal state of the object cannot be changed. There are many issues concerning representation and invocation of database behaviour, but they will not be addressed here as they form part of a separate ongoing project[10].

### 4.6 Defining Schemas

![Figure 7: Metadata Definition of Schema.](image)

A schema represents the top level container for classes and object views. The ODMG implements a schema as an instance of the `d_Module` metaclass, while in our metamodel `sys_Schema`, derived from `sys_ScopedObject`, is an abstraction for database schema and view subschema metaclasses. An instance of `sys_DatabaseSchema` represents one database schema,
defining a global scope for the database objects it contains. Objects that can be registered within a database schema are base classes and views.

4.7 Defining Views

The `sys_SubSchema` is a special “schema” for representing object views. It contains base and virtual classes and each subschema belongs to one database schema. This representation is necessary as a view is always based on a subschema, and not on a single virtual class. Object views provide schema restructuring functionality for object-oriented database models. This feature is commonly used in federated database systems for the construction of different component and federated schemas. Object view support is not provided in the ODMG standard, but our metamodel defines extensions for representing view metadata. View support in this metamodel is designed to represent the view mechanism specified in [21]. Each object view, represented as an instance of `sys_SubSchema` metaclass, contains one or more virtual classes. Virtual class is recursively constructed from base classes or other virtual classes using restructuring operators specified in [21].

4.8 A Meta-Metadata Level

Each metadata model describes the structure of the database schema at some level of abstraction. Our metadata model is specifically constructed to support multimedia metadata by recognising multimedia types as a special form of data type. The model in which the metamodel is specified and constructed is called the meta-metadata model. Metadata models for representation of specific database models (e.g. multimedia) can be easily defined in the meta-metadata model. Migration from the one metamodel structure to the another is accomplished by changing metamodel representation in the meta-metadata model. Our specification of meta-metadata model is illustrated in Figure 8. We recognise the meta-metadata model as beneficial to our system, because it will allow us to specify new metamodels and to add new metaclasses to the existing ones.

The `m_Abstract` metaclass is an abstraction for any type of metamodel element. It defines `name` and `comment` attributes common for all entities in the meta-metadata model. The `m_Abstract` can be realised as element, attribute, association, generalisation or schema. The `m_Element` metaclass represents a general container element, and instances of the `m_Element` correspond to `sys_Class` metaclasses in the metamodel definition. Each `m_Element` can contain attributes and associations represented by the `m_Attribute` and `m_Association`. The `type` defines attribute type, while `isUnique` property specifies if attribute instances must be unique. Associations in the meta-metadata model can be unidirectional (`m_UniAssociation`) or bidirectional (`m_BiAssociation`). The cardinality, `isUnique` and `isOrdered` properties are defined for both association types, but only bidirectional associations have a traversal link to the inverse association element. Inheritance relationships between metadata elements are represented by the `m_Generalisation` metaclass, while the `m_Schema` is the root container for all elements in the meta-metadata model. Each `m_Schema` instance corresponds to one metamodel schema defined in the meta-metadata model. The `associable_elements` relationship between `m_Association` and `m_Element` metaclasses defines association rules for metamodel instances of the meta-metadata model. The relationship specifies all subclasses to which an association defined between their superclasses, can be propagated. This feature enables definition of strict association rules that specify which subclasses are allowed to create an association link defined for their superclasses.

5 Metamodel Mappings

Mappings are defined as a set of rules for transforming the EGTV metamodel to the object-relational and ODMG object-oriented metamodel representations.
Mappings are generally straightforward except for some features of the EGTV metamodel not included in the ODMG and object-relational specifications such as multimedia types and object views. The ODMG and object relational metamodels must be extended to support mappings to new EGTV features. For simplicity in this discussion, we represent all ODMG and object-relational metamodel extensions either as a single ODMG class \(d_\text{Extension}\) or as an object-relational table \(\text{all_extensions}\). For a full description of the ODMG object-oriented standard, please refer to the [6], the object-relational SQL99 features are explained in the [7], while EGTV metamodel extensions are specified in [1].

5.1 EGTV To ODMG Mapping

The EGTV metamodel is based on the ODMG metamodel specification, so both metamodels share a similar object-oriented platform. The mapping is relatively simple for metaclasses that have similar definition in both metamodels, but is more complex for multimedia types and object views. This section explains mapping rules, while major EGTV metaclasses and their mappings to the ODMG metamodel are illustrated in Table 1. In some of these descriptions we provide a formal mapping language example to demonstrate. For a full list of mapping commands for each metaclass, please refer to [3].

<table>
<thead>
<tr>
<th>EGTV metaclass</th>
<th>ODMG metaclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_DatabaseSchema</td>
<td>d_Module</td>
</tr>
<tr>
<td>sys_Class</td>
<td>d_Class</td>
</tr>
<tr>
<td>sys_Attribute</td>
<td>d_Attribute</td>
</tr>
<tr>
<td>sys_Inheritance</td>
<td>d_Inheritance</td>
</tr>
<tr>
<td>sys_Relationship</td>
<td>d_Relationship</td>
</tr>
<tr>
<td>sys_Method, sys_Operator</td>
<td>d_Operation</td>
</tr>
<tr>
<td>sys_Parameter</td>
<td>d_Parameter</td>
</tr>
<tr>
<td>sys_PrimitiveType</td>
<td>d_Primitive_Type</td>
</tr>
<tr>
<td>sys_CollectionType</td>
<td>d_Collection_Type</td>
</tr>
<tr>
<td>sys_MediaType</td>
<td>d_Class</td>
</tr>
<tr>
<td>sys_SubSchema</td>
<td>Extended ODMG</td>
</tr>
</tbody>
</table>

Table 1: The EGTV to ODMG Mapping.

5.1.1 Schema Mapping

The \textit{sys_DatabaseSchema} metaclass defines database schema properties including a root naming scope for all schema elements. It is mapped to the \textit{d_Module} metaclass which represents the equivalent class in the ODMG metamodel. For each database schema specified in the \textit{sys_DatabaseSchema}, the mapping creates one \textit{d_Module} instance in the ODMG metamodel. The schema mapping is illustrated in Example 1.

```plaintext
map sys_DatabaseSchema := d_Module, d_Extension |
  attribute:
    isGlobal := d_Extension.isGlobal
  databaseType := d_Extension.databaseType
  relationship:
    containedIn := d_Module.definedIn
    containedObjects := d_Module.defines
```

Example 1: ODMG Schema Mapping.

5.1.2 Class Mapping

Both metamodels represent database classes through a single metaclass. This mapping is illustrated in Example 2.

```plaintext
map sys_Class := d_Class, d_Extension |
  attribute:
    isAbstract := d_Extension.isAbstract
  relationship:
    inheritsTo := d_Class.inherits
    derivesFrom := d_Class.derives
```

Example 2: ODMG Class Mapping.

The map function defines the mapping between EGTV \textit{sys_Class} and ODMG \textit{d_Class} metaclasses. Properties of \textit{sys_Class} for which equivalents can not
be found in the d_Class are mapped to the ODMG extensions (d_Extension). Attributes having the same name and type in both metamodels (name and comment) are implicit. The isAbstract attribute is new to EGTV metamodel, and maps to the same attribute in the d_Extension class. The containedObjects EGTV relationship represents attributes, operations and relationships contained within the instance of the sys_Cass, and maps to the definedIn relationship of the d_Class. The class containment in the database schema is represented with the containedIn relationship which maps to the defines relationship of the d_class. The inheritance relationships inheritsTo and derivesFrom map to the d_Class inheritance relationship inherits and derives. The EGTV sys_Class defines additional attributes virtualLevel, operatorType and relationship virtualConnector for representing virtual classes. These attributes are mapped only to the the ODMG extensions for object views [21].

5.1.3 Attribute Mapping

Each attribute defined in the EGTV sys_Attribute class is mapped to one instance of ODMG d_Attribute metaclass. All attributes and relationships of the sys_Attribute are mapped to the corresponding attributes and relationships of the d_Attribute.

5.1.4 Inheritance Mapping

Inheritance mapping is from the sys_Inheritance metaclass to the ODMG defined class d_Inheritance. All attributes of sys_Inheritance are mapped to the d_Inheritance attributes, while the inheritsTo and derivesFrom relationships between sys_Inheritance and sys_Class are mapped to the same relationships (inherits and derives) between d_Inheritance and d_Class.

5.1.5 Relationship Mapping

The EGTV sys_Relationship metaclass maps to the ODMG metaclass d_Relationship. Attributes in the sys_Relationship which do not have an equivalents in d_Relationship, are mapped to the ODMG extensions. The relationship mapping is illustrated in Example 3.

The map function maps the sys_Relationship EGTV metaclass to the d_Relationship and d_Extension classes. The name, comment, cardinality and traversal properties of sys_Relationship are not specified in Example 3 as these mappings are implicit in this function. The isUnique and isOrdered attributes do not have counterparts in the d_Relationship, and are mapped to the ODMG extension class.

5.1.6 Data type mapping

Primitive types and collection types are represented in EGTV in the same way as in the ODMG metamodel. The sys_PrimitiveType is mapped to the d_Primitive_Type and sys_CollectionType is mapped to the d_Collection_Type. This mapping is defined in Table 2. Multimedia types were introduced in the EGTV metamodel and do not have counterparts in the ODMG. Furthermore, multimedia types can define behaviour operations, which is not supported for types in ODMG metamodel. The solution is to map the EGTV sys_MediaType to d_Class in the ODMG specification. This means that EGTV multimedia data types are represented in the ODMG metamodel as system defined classes. For example the jpgImage class in the d_Class meta-class. The names of the multimedia classes are system reserved and users cannot define classes of the same name. Since classes in the d_Class can have operations, the multimedia type operations are mapped to operations of the ODMG classes, however due to limitations of ODMG, interface mapping only is provided, and not actual behaviour.

5.1.7 Object View Mapping

Object views consist of one or more virtual classes and are represented as a view subschema as defined in our previous work [21]. The EGTV metamodel represents object views and virtual classes in the same set of metaclasses used for schema and base classes [1].

Example 3: ODMG Relationship Mapping.

\[
\text{map \ sys\_Relationship := d\_Relationship, d\_Extension} \\
\{ \\
\text{attribute:} \\
\text{cardinality := d\_Relationship\_cardinality} \\
\text{accessKind := d\_Relationship\_accessKind} \\
\text{isUnique := d\_Extension\_isUnique} \\
\text{isOrdered := d\_Extension\_isOrdered} \\
\text{relationship:} \\
\text{containedIn := d\_Relationship\_definedIn} \\
\text{traversal := d\_Relationship\_traversal} \\
\} \\
\]
The ODMG metamodel does not provide support for object views, so they can be mapped only to the extended ODMG metamodel specified in [22]. The mapping rules for object views are not discussed here as they are part of our future research in federated layer of EGTV architecture.

<table>
<thead>
<tr>
<th>EGTV Type</th>
<th>ODMG Type</th>
<th>O-R Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>long</td>
<td>integer</td>
</tr>
<tr>
<td>Float</td>
<td>float</td>
<td>number</td>
</tr>
<tr>
<td>String</td>
<td>string</td>
<td>varchar</td>
</tr>
<tr>
<td>Date</td>
<td>timestamp</td>
<td>date</td>
</tr>
<tr>
<td>Boolean</td>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>Bag</td>
<td>Bag</td>
<td>nested table</td>
</tr>
<tr>
<td>List</td>
<td>List</td>
<td>array</td>
</tr>
</tbody>
</table>

Table 2: EGTV Type Mappings

5.2 EGTV To Object-Relational Mapping

Object-relational databases do not have a standardised metamodel, so in this discussion we will regard the Oracle 9i metamodel as a standard as it has the most advanced features. The object-relational schema repository is represented as a set of relational tables which store both relational and object-relational metadata [24]. The EGTV metaclasses are mapped to the Oracle 9i schema repository tables as illustrated in Table 3. The attributes of the EGTV metaclasses are mapped to the table columns, while relationships are mapped to referential integrity constraints between schema repository tables. Some examples are given here, but a full list is provided in [3].

<table>
<thead>
<tr>
<th>EGTV Metaclass</th>
<th>O-R Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_DatabaseSchema</td>
<td>all_users</td>
</tr>
<tr>
<td>sys_Class</td>
<td>all_types, all_object_tables</td>
</tr>
<tr>
<td>sys_Attribute</td>
<td>all_type_attrs, all_tab_columns</td>
</tr>
<tr>
<td>sys_Inheritance</td>
<td>all_object_types</td>
</tr>
<tr>
<td>sys_Relationship</td>
<td>all_type_atts</td>
</tr>
<tr>
<td>sys_Method, sys_Operator</td>
<td>all_type_methods</td>
</tr>
<tr>
<td>sys_PrimitiveType</td>
<td>all_types</td>
</tr>
<tr>
<td>sys_MediaType</td>
<td>all_object_types</td>
</tr>
<tr>
<td>sys_SubSchema</td>
<td>Extended O-R</td>
</tr>
</tbody>
</table>

Table 3: EGTV to Object-Relational Mapping

5.2.1 Schema Mapping

In the object-relational model, each database user owns one database schema which contains object types, tables and other model elements. The object relational schema repository represents database schemas in the all_users system table to which the EGTV sys_DatabaseSchema metaclass is mapped. The mapping represents each schema defined in the sys_DatabaseSchema as one tuple of the all_users table. Each schema defined in the all_users table provide root naming scope for all types and tables it contains. This is illustrated in Example 4.

map sys_DatabaseSchema := all_users, all_extensions |
  attribute:
    name := all_users.username
    comment := all_extensions.comment
    isGlobal := all_extensions.isGlobal
    databaseType := all_extensions.databaseType
  relationship:
    containedObjects := all_users.username ref_to
    all_types.owner
}

Example 4: O-R Schema Mapping.

The map function maps the sys_DatabaseSchema EGTV metaclass to the O-R table all_users. The name attribute is mapped to the username column of the all_users table. The other attributes of the sys_DatabaseSchema do not have counterparts in the all_users table, and are mapped to the O-R metamodel extensions (all_extensions). The relationship containedObjects between sys_DatabaseSchema and sys_Class metaclasses is mapped to the referential integrity constraint between all_users and all_types tables.

5.2.2 Class Mapping

Classes in the object-relational model are represented as user defined types (UDT) and instantiated in the form of object tables [17]. For this reason, the sys_Class from the EGTV metamodel is mapped to two O-R schema repository tables: all_types and all_object_tables. The all_types table contains all user defined types in the database schema, while the all_object_tables represent object tables that instantiate these UDTs. The class mapping is illustrated in Example 5.
map sys_Class := all_types, all_object_tables, all_extensions

{ attribute:
    name := all_types.type_name, all_object_tables.table_name
    comment := all_extensions.comment
    isAbstract := all_types.instantiable
    relationship:
        containedIn := all_users.username ref_to
                      all_types.owner
        containedObjects := all_types.type_name ref_to
                        all_type_attrs.type_name,
                        all_types.type_name ref_to
                        all_type_methods.type_name
        derivesFrom := all_types.supertype_name
        inheritsTo := all_extensions.inheritsTo
    }

Example 5: O-R Class Mapping.

The object-relational mapping requires a map function which transforms the sys_Class metaclass to all_object_tables and all_types O-R tables. The name attribute is mapped to name columns of both all_types and all_object_tables tables, while the comment attribute (not existing in the O-R schema repository) is mapped to the all_extensions table. The isAbstract attribute maps to the instantiable column of the all_types table and containedIn relationship between class and its containing schema is mapped to referential integrity constraint between all_types and all_users tables. Attributes and relationships defined within the class (containedObjects relationship) are mapped to referential integrity constraint between all_types and all_type_attrs tables, while operations are mapped to the constraint between all_types and all_type_methods tables. Inheritance in the sys_Class is represented with derivesFrom and inheritsTo relationships. The first relationship is mapped to the supertype_name column of the all_types table, while the second one maps to the O-R schema repository extensions.

5.2.3 Attribute Mapping

Class attributes defined in the EGTV metaclass sys_Attribute are mapped to two object-relational tables. The all_type_attrs table defines UDT attributes, while all_tab_columns stores their object table instantiations. The containment relationship between attributes and user defined types is mapped to the referential integrity constraint between all_type_attrs and all_types schema repository tables. This is illustrated in Example 5.

5.2.4 Inheritance Mapping

Inheritance in the object-relational metamodel is represented as relationship between subclass and a super-class instances of the all_types schema repository table. The sys_Inheritance EGTV metaclass maps to the inheritance property supertype_name of this table.

5.2.5 Relationship Mapping

Relationships are represented as reference (REF) attributes and nested tables (collections) of REF attribute types [17]. The REF attribute type represents one side of the relationship, while the nested table of REF types is used to describe many side of the relationship. The object-relational mapping translates relationships from the EGTV sys_Relationship metaclass to the all_type_attr table. Since the class attributes are also mapped to the same table, the property attr_type of this table is used to distinguish relationships (REF attributes) from the non-reference attributes.

5.2.6 Data Type Mapping

Primitive and collection data types are mapped from the sys_PrimitiveType and sys_CollectionType EGTV metaclasses to the all_types table in the object-relational schema repository. The mappings are defined in Table 2. Multimedia types are not defined in the O-R schema repository, so they are represented as a special user defined types in the all_object_types table. For example, the mpeg multimedia data type defined in the sys_MultimediaType EGTV metaclass is mapped to the mpegVideo user defined type in the all_object_types table. Operations defined for multimedia types are mapped to UDT operations. Regular UDTs cannot have the same name as these UDTs for multimedia representation, since these names are reserved for multimedia types representation.

5.2.7 Object View mapping

The object-relational metamodel supports only a limited form of views. For this reason the EGTV view subschemas must be mapped to the object-relational metamodel extensions. These metamodel extensions include a set of tables which supplement the existing object-relational schema repository with the view metadata.
We defer discussion on view mapping since it forms part of current research.

5.3 Schema Construction

EGTV multimedia database schemas can be constructed either in bottom-up or top-down approach.

**Bottom-up.** Database schemas are first specified in the native Data Definition Language (DDL) of object-oriented or object-relational database. These DDL definitions are then processed to create classes, properties and other metadata elements in the database schema repository. Finally, metamodel mapping rules are applied to transform these schema definitions from the proprietary database metamodel to the EGTV metamodel representation.

**Top-down.** Database schema metadata is first created in the EGTV metamodel representation using platform independent data definition language ODLx [2]. This language modifies ODMG ODL [6] providing EGTV metamodel support and XML encoding. Metamodel mapping rules are then applied to transform metadata from the EGTV representation to the object-oriented or object-relational database metamodel, and to create database schemas.

6 Implementation

A working prototype has been implemented for Versant 6.0 (ODMG) and Oracle 9i (object-relational) databases. It was developed and tested on Windows platform (NT/2000/XP), and Linux (RedHat 7.2). The metamodel implementation for Versant is specified in C++ headers, stored in the database using the Versant schema import utility. This process creates an extended schema repository which implements metamodel in a form of persistent C++ classes and relationships between them. The Oracle implementation of metamodel is developed as a SQL-99 script file containing SQL DDL statements for creating object types and object tables. This SQL script is executed to create schema repository implemented as a set of object-relational tables constructed from object type definitions.

The existing test site is developed to support Fischlãár video system, and it contains three autonomous multimedia databases extended with the EGTV metamodel [16]. The first database defines schema for television recording system. The system receives daily television schedules (both in textual and video format) and enables registered users to select programs to be recorded. This information is stored in the Versant ODMG database and it can be accessed through the EGTV metamodel. The second database is video data store for recorded television programs. It uses Oracle database server for storage of indexed MPEG video files. Each recorded video contains additional information that describes its content. This information can be categorised by the genre to which the video belongs. For example each film will have information about actors and director, while news can be classified by TV station and presenters. The video editing system is a third segment of our test site. It defines the database schema for creating complex multimedia presentations from existing audio, video and hyper-text multimedia contents. This schema is located in the Versant database server. Each of these schemas was first defined in the EGTV metamodel representation and then mapped to Oracle and Versant databases in the top-down approach. By implementing the the EGTV metamodel we are now in the position to interrogate schemas and query data in the platform independent way, regardless of database type. The next step is to integrate these three databases to a federation. Federated schema is constructed from a set of views [21] that span multiple multimedia databases. This will enable us to issue global queries against data contained in multiple sources.

7 Conclusions

In this paper we described our approach to designing and implementing a metamodel for multimedia databases. The metamodel defines a set of metaclasses for metadata storage in our multimedia database federation. This metamodel is based on the ODMG object-oriented metamodel, but provides a more simplified design, multimedia data types and extended support for object views.

So far, we have developed and implemented a standard metamodel interface to object-based multimedia systems, while the contribution is in the fact that no federated database research project (covered by this group) has published their metamodel. By publishing the metamodel, it is possible to create an architecture which is both open and extensible. We feel that this will prove to be beneficial when creating federations of multimedia databases, which forms part of our current research focus. The query language for multimedia database
schemata is an issue which also need to be addressed in the future research. This language should be capable of querying schemata at both canonical and federated layers of our architecture.

References


