Interoperability and Multimedia Archives

Dalen Kambur  Damir Bećarević  Mark Roantree
Interoperable Systems Group, Dublin City University, Dublin 9, IRELAND
{dalen.kambur, damir.becarevic, mark.roantree}@computing.dcu.ie

Abstract

In distributed computing systems, it is unwise to move data to the point of program code, but instead process data at the point of storage. This concept is even more appropriate to multimedia repositories where large files must be processed as a result of each retrieval operation. Two problems with multimedia databases are that they require a large volume of disk storage and processing power and that it is generally difficult to query video content or optimise the retrieval process. In the EGTV project, both of these issues are addressed. In the first case, the data server is distributed across multiple autonomous sites where users are permitted to modify schemas and in some cases the data model. In the second case, a mechanism for storing behaviour has been devised which allows functions such as retrieve_frame, retrieve_segment and retrieve_context to be implemented and stored at individual servers. This provides selective retrieval and thus, large data items can be processed locally to filter unwanted data before the transfer process must begin. In this way, storage of operations forms the basis for optimisation of retrieval content and volumes.

1. Introduction

The EGTV project [10] was formed with the goals of distributing multimedia stores onto smaller and cheaper systems while at the same time providing a means of federating these smaller multimedia databases. Its goal was to allow ordinary users to create on-line archives using the model and schema design of their choice. It was also aimed at providing a federation of digital archives for organisations such as broadcasting stations, healthcare and libraries. A federated approach was required to enable users to reconfigure schemas, functional requirements, archive processing and video editing capabilities. One key requirement was that different user groups may choose to change the level at which they see the database: engineers may wish to provide new data types and operations for the latest multimedia files (MPEG-7 for example), whereas some end-users (e.g. media organisations) have a requirement to archive data in such a way that it is easily retrievable using some form of query capability. Thus, the process of periodically archiving to DVD is not a realistic option. The Físchlár Digital Multimedia System [8] allows over a thousand users to process, record and playback television programmes using an Oracle Video Server platform. However, its centralised architecture restricts its processing power and storage capabilities, thus limiting the number of concurrent users and volume of storable (and queryable) multimedia content. Furthermore, it was felt that a centralised system operating with a homogeneous set of rules and requirements had become too restrictive given the wide ranging needs and hardware capabilities of each group of users. Thus, it provided a suitable test domain for the research undertaken in the EGTV project.

The client applications interact with the EGTV software services through an EGTV database interface, wrapped in programming language friendly class libraries. The interface to the class libraries is similar to the ODMG [3] object manipulation interface (Object Manipulation Language) in providing an object-oriented access to stored objects. The ODMG’s “look and feel” was chosen as the object-relational standard does not provide such libraries. The EGTV interface (compared to the ODMG one) is enriched with behaviour storing and updateable views. To achieve these enrichments, the internal structure of the model has been completely re-specified with the strong influence of the LOQIS [13] database model. Our model provides a stricter query language semantics (the result of a query is a set of objects) than OQL, which in turn facilitates updateable query result and clean query nesting. In this paper, we explain how behaviour is defined and used in queries in the EGTV project.

Contribution. In summary, the contribution of this paper is a mechanism for storing multimedia operations in object databases that are acting as multimedia archives. We do not
address issues of streaming but the local processing and retrieval of multimedia objects. This is achieved by providing an ODMG-type model with clearer semantics and providing a more functional or reference-based methodology to object types. In the EGTV project, this stored behaviour is used to facilitate queries and provide a form of optimisation as there is no need to ship large volumes of data to code, rather the code is located with the data, and processing occurs locally. The paper is structured as follows: the EGTV model and semantics are covered in §2 with the behavioural aspects of the model in §3; The query language is described in §4; and finally, §5 concludes the paper. Please refer to [5] for full version of this paper which includes a discussion on related research.

2. Using the EGTV Model

In this section, a brief introduction to the EGTV model is provided in §2.1. A discussion on transparent object materialisation is then given in §2.2.

The model and behaviour are introduced using a schema definition process of the Film Archive Schema, illustrated in Figure 1. The schema is defined using the ODLx schema definition language [1] and consists of three classes: Film, MotionPicture and Actor which are part of an existing database schema. Using this schema, a view consisting of a single virtual class RecentFilms is defined.

![Figure 1. Film Archive Schema.](image)

2.1. EGTV Model Overview

The EGTV model [6] is an object model extended with the concept of references. Objects, properties, classes and behaviour are defined similar to other object models while a reference points to an object or object property in an uniform manner. This is achieved by giving unique identifiers both to objects and their properties, referred to here as egtvOIDs (egtv Object IDentifiers) and similar to the identification techniques used in [14]. This is different to traditional object models, where an object is identified using its unique OID (Object IDentifier), and a property is then identified using a pair (OID, propertyName). However, as shown in [12], this leads to ambiguities where the second part of the pair (propertyName) contains multiple values.

An egtv object does not contain, rather it aggregates its properties using references. Thus its properties need not be consecutively stored, similar to the object-slicing technique introduced in the MultiView project [11]. This feature also influenced the semantics of the query languages which is also part of the model definition. Properties are of one of the built-in types, where a built-in type defines a set of valid states for a property and behaviour used to modify the state of the property. An EGTV class description defines the names and types for each of the properties. Each property is an attribute or a relationship, where relationships are specified in pairs of relationship-sides and bi-directional (a modification of one side of the relationship is reflected to its traversal to maintain the referential integrity). Internally, a relationship-side encapsulates one, a set or a list of references implementing traditional one-to, unordered many-to and ordered many-to relationships. No other means of addressing objects or properties is provided in the model except of references. For a full description of the EGTV model, please refer to [6].

**Behaviour.** The behaviour of a class is defined as a set of operations. An operation is a part of the class definition and is defined against a target object and the parameters the operation receives. For each parameter, its mutability and type is specified. The operation’s implementation manipulates these to evaluate the return value for which the type needs to be specified. The target object, parameters and the return value are communicated using references, thus each of these can be an egtv object or its property. This feature of the model forms a large part of the discussion in §3.

**Semantics of Query Languages.** The query languages defined for the EGTV model must adopt the EGTV query language semantics. In §4 we introduce EQL [2], a fully functional query language that was based on the EGTV model.

An EGTV query specifies the transformations to be applied against source egtv objects. To represent the result, each successful transformation is materialised as a separate virtual egtv object, employing object-generating semantics [7]. These aggregate their properties in the same way as base egtv objects and appear no different to client applications. Additionally, this provides an orthogonality between the query input and the result as both are sets of egtv objects,
resulting in a clear theoretical basis for nesting of queries within queries.

The transformations are expressed in terms of source, selection and projection clause. Where the source clause specifies the source objects (objects to be transformed), the selection clause specifies the criteria of successful transformation (criteria all virtual objects must fulfil) and finally, the projection clause specifies the properties the virtual objects will have. Properties of virtual objects can be projected from base objects or be constructed using an operation invocation. While creation of virtual objects results in a new OID for the virtual object, the OIDs of properties remain the same, thus employing object-preserving semantics. This provides updateability of the virtual object and when virtual properties are updated, these updates are persistent. The sole exception is in the case of newly constructed properties (as a result of an operation) which are read-only. The materialised virtual objects belong to the generated virtual class.

To summarise, object-generating semantics is employed for virtual objects while object-preserving one is employed for their properties. This approach was chosen as it is compatible with current object-oriented languages (i.e. does not require an object to belong to multiple classes), while allowing for updateability of virtual objects. As base and virtual objects have the same format (i.e. reference their properties), it is possible to define new behaviour in virtual classes.

2.2. Object Materialisation

The client applications and class operations address objects using only references. As manipulation is restricted to egtv objects only, persistent objects must be materialised before use. The references provide transparent materialisation-on-demand, a process resulting in importing a persistent object into the EGTV Object Pool. The object can originate from ODMG and O-R [9] databases, but also from another EGTV database when used in a federated database environment. Thus, the object can be reimported across several nodes and additional behaviour defined. This facilitates the usage of the EGTV model as a grid-computing model. The egtv object is automatically closed when the last reference pointing to it is removed. This is explained in detail in §4.1. Using this simple approach, we eliminate the problem of maintaining identifiers present in object-generating-semantics as the virtual objects are closed after closing their last reference.

3. Defining Class Behaviour

A class definition consists of structure and behaviour. The structure determines the names and types of properties and is defined differently for base and virtual classes. In the case of base classes, the names and types of properties are specified and used to create a schema in the Data Repository. In the case of virtual classes, the query that specifies the virtual class extent is used to deduce the structure of the class. The metadata descriptions are stored within the Schema Repository.

An operation definition consists of two parts: a declaration and an implementation.

Declaration. An operation’s declaration is used to generate a metadata description of the operation, stored within the Schema Repository. The declaration consists of the class identification, operation’s signature and return type. The operation’s signature, consisting of parameter type declarations is unique within a class and is associated with that class.

Implementation. To implement an operation, the library in which the operation is stored is specified and the operation’s source code. The implementation is stored in the Behaviour Repository, which takes a form of a collection of EGTV libraries. Libraries are generated from operation definitions in two stages. In the first stage, the library source code is generated, consisting of operation signatures, their implementation and the registration code, all expressed using a library implementation programming language. The implementation is similar for the base and virtual classes and is given shortly. The registration code registers all operations defined in the library with the EGTV database system. In the second stage, the library source code is then built using standard tools (compiler/linker) into a library and stored with the Behaviour Repository.

To implement the operation’s functionality, a base class operation’s source code uses the EGTV class library to manipulate the target egtv object and its properties. This library wraps the EGTV object algebra and represents it in a programming language friendly manner. In addition to properties, a virtual class operation manipulates source objects used to construct the virtual one. These appear as additional attributes of the virtual object. The EGTV model does not facilitate importing a source operation into a view as the source operation was defined for the source class and applying such an operation on an object of transformed class is inappropriate. However, the view designer can define an operation with the same signature which invokes a source class operation. This simulates the import functionality for operations. As source operation’s implementation is not available (it is a black-box), the EGTV system performs no checking whether the virtual class contains all the properties that the base operation manipulates with. The processing of the behaviour in views is not different to processing of the class behaviour due to the reference nature of the EGTV model.
3.1. Default Operations

The definition process generates default operations for each class to simplify introducing new classes. For base classes, the following default operations are provided: a constructor copies the received parameters into the object’s properties; an equality operator tests two objects for their equality, property by property; an assignment operator assigns the values of properties of one object to the other one, property by property; and finally, a destructor closes the object’s properties and the object itself.

The only default operations generated for views are the equality and assignment operators. The constructor and the destructor are not generated as their semantics cannot be inferred from the view specification. For instance, from the definition of the view RecentFilms, it cannot be inferred what a constructor is meant to do when a client application tries to create a very old film. Thus, to facilitate creation and deletion of virtual objects of the view the constructor and destructor methods must specify how the virtual object creation or deletion affects the source objects.

The class designer can customise each of these default implementations by specifying an operation with the signature corresponding to the operation’s signature. For example, the default equality operator comparing two objects of MotionPictures class would compare their attributes for equality. However, the time-consuming comparison of the media attributes which stores the multimedia material, might not be necessary to establish equality of two MotionPictures (in fact, it may be even dubious considering the stored material might contain different broadcaster logos) so the class designer can override the equality operator to compare only name, year and director attributes.

4. Using Behaviour in EQL

The EGTV Query Language (EQL) provides querying and updating for (distributed) EGTV multimedia database schemas and it follows the query language semantics introduced in §2.1. A full description of the query language is provided in [5, 2], where we here illustrate its usage with behaviour. The query in Example 1 retrieves a list of years, lengths and actors of short films directed by Henry Lehrman. This query uses method isShortFilm to determine if the film is short, and operator = on property director to determine if the film was directed by Henry Lehrman.

4.1. Invoking Behaviour from Queries

A typical query language defines manipulations with extents and objects, despite the fact that the object manipulations are already a part of the type definition of the model. As opposed to this, the semantics of EGTV query languages restricts a query language from providing object manipulations. Instead, the query language defines class manipulations and invokes the existing defined operations on objects to achieve object manipulation. Thus, a Query Processor is a generic application, manipulating classes solely through their operations and has no prior knowledge of the available operations and state except of the information retrieved from the metadata. A typical example of such a double definition is the logical and operator. It is usually defined as a part of the query language despite the fact that the underlying data model provides both the boolean data type and associated and operator. In the EGTV model we remove such duplicate definitions.

An EQL query given in Example 1 retrieves the year, length and actors of short films directed by Henry Lehrman. The execution of this query is illustrated in Figure 2.

```
select year, length() as length, m.cast.name as actor
from MotionPicture m connect Actor
on m.cast
where director="Henry Lehrman" and isShortFilm();
```

Example 1. Navigational join.

Figure 2. Operational Architecture.

**Operational Architecture Description.** The Data Repository and the Schema Repository are implemented either as an O-R or an ODMG database. The Behaviour Repository is realised as a folder on the file-system containing dynamic libraries as separate files. The Object Manager consists of the Database Accessor and Behaviour Accessor, and is responsible for maintaining the EGTV Object
Pool. All egtv base and virtual objects (shown in rounded boxes) are part of the pool and it is EGTV’s equivalent of the traditional database cache. The triangles denote egtvOIDs associated with the particular object/property. We omitted their value to simplify the illustration. The dashed arrow between the year properties denotes it is the same property, just aggregated both from a base and a virtual object. The same applies for the dashed arrow connecting actor and name properties. The relationship between MotionPicture and Actor (film-cast) is omitted to simplify the illustration.

The Database Accessor abstracts the database dependencies by providing an uniform interface to stored data and metadata. Thus, the egtv base objects are materialised and manipulated through the Database Accessor. The Behaviour Accessor loads the libraries needed for a particular operation implementation and it natively provides the functionality of the built-in type operations. Virtual egtv objects are materialised by the Query Processor and they are used to represent the result of a query execution, the details of which are provided later. This internal structure is encapsulated from client applications using a Session object which implements the EGTV server interface and is instantiated for each client application requesting a connection. The client application forwards all communication with the EGTV database through this object. Each Session object has its own EGTV Object Pool. The object pool is empty at the moment the connection is established. The Session object operates with both base and virtual egtv objects using the same interface and not knowing whether they are base or virtual.

Query Evaluation. The source objects must be transformed to the virtual ones aggregating projected properties, where the selection clause must be satisfied. The selection clause is an expression, evaluating to a boolean property. When it evaluates to true, the transformation is considered to be successful and a virtual object is created.

In our sample query, it is necessary to determine if the film is a short one and directed by Henry Lehrman. To determine if the film was directed by Henry Lehrman, the operator equals is invoked against property director and string parameter “Henry Lehrman”. To determine if it is a short film, the operation isShortFilm is used. To combine both of these, the operator and belonging to the built-in type boolean is used as (different to other query languages) the query language does not provide a logical and operator. Though this appears to be a restriction, it is an advantage as the query language does not redefine the operations provided by the underlying data model.

An execution of the isShortFilm operation invokes the length operator against the media property to determine if the multimedia material complies with the requirement for a short film (illustrated with an arrow “f”).

While the property year results from projecting the property year from the base class, the property length results from projecting the result of the operation length as a property (arrow “e”). This property maintains the value acquired at the moment of evaluation, and to re-evaluate it, the query needs to be resubmitted. Finally, the property actor results from the navigational join with the Actor class.

Submitting a Query. The client application issues the sample query against the Session object. The query is then forwarded to the Query Processor (arrow “a”). The Query Processor analyses the query and determines that the extent of the class MotionPicture is needed for the query evaluation.

Class Materialisation. This request is forwarded to the Object Manager. The request for the class extent begins with retrieving the class metadata (arrow “d”) needed to materialise base objects. The metadata also contains the list of libraries used by the operations of the class. This list is forwarded to the Behaviour Accessor to load the libraries needed by the class operations. Then the Database Accessor materialises the MotionPicture extent. This extent is of type set and contains references to objects of the class MotionPicture. The objects themselves are not materialised. The extent is returned to the Query Processor.

Processing Query Predicates. The source extent needs to be filtered using the selection clause. The process of filtering is applied against the extent, the objects of which need to be materialised first (arrow “c”), as in the previous step, only the references were retrieved from the database.

Materialising Virtual Objects. The query processor uses the filtered source extent to materialise the extent of virtual egtv objects, as illustrated with arrow “b”. The materialised virtual objects contain two properties, one of which is the year property of source objects. The other one is evaluated by invoking the method length on the source extent (arrow “e”).

5. Conclusions and Further Research

In this paper we presented the EGTV object database model, the mechanism used for specifying and storing behaviour, the semantics which demonstrate how queries are processed, and thus, a platform for storing multimedia operations at local databases. This work provides a contribution in its provision for video engineers to create and store query and retrieval functions at local databases and thus, removing
(or at least reducing) the need to move large volumes of data to the location of the processing program code. The EGTV model has also demonstrated that it can be applied in a federation of data stores: it provides mappings to both ODMG and O-R databases, an interoperable layer, and the query language functionality for global queries through the EQL. Finally, the reference based model and object-preserving semantics for object properties guarantee updates through remote object interfaces.

A demonstration prototype was built as a typical client-server architecture. Client applications access the EGTV server through a CORBA bridge, implemented using Iona’s CORBA engine [4]. CORBA facilities sending messages to remote objects in a platform and programming language independent manner, thus providing a communication interface between EGTV database nodes. As CORBA is a binary (non-textual) communication standard, it is used for management of large textual and multimedia data. This communication interface is wrapped inside EGTV system classes, which provide a programming language friendly interface. This was implemented using C++ on a Windows platform.

The EGTV server is based on Oracle 9i and Versant databases on both Windows XP and Linux platforms. Users designed five separate database schemas according to their different goals and design patterns: multimedia recording system, multimedia editing system, TV news archive, sports database, and Irish language news database. The multimedia content was mainly drawn from the Físchlár [8] system, although once stored locally, it was possible for users to manipulate the data set according to their own needs.

References