

# AN OBJECT MODEL FOR THE MANAGEMENT OF DIGITAL IMAGES

S. Khaddaj and A. Hoppe

*Faculty of Computing, Information System and Mathematics, Kingston University  
Kingston upon Thames KT1 2EE, U.K.*

**Keywords:** Object Oriented Modelling, Object Versioning, Image Management Systems.

**Abstract:** With digital image volumes rising dramatically there exists an important and urgent need for novel techniques and mechanisms that provide efficient storage and retrieval facilities of the voluminous data generated daily. It is already widely accepted that the use of data abstraction in object oriented modelling enables real world objects to be well represented in information systems. In this work we are particularly interested with the use of object oriented techniques for the management of digital images. Object orientation is well suited for such systems, which require the ability to handle multiple type content. This paper aims to investigate a conceptual model, based on object versioning techniques, which will represent the semantics in order to allow the continuity and pattern of changes of images to be determined over time.

## 1 INTRODUCTION

Digital imaging management systems, in which images are collected, organised and categorised to facilitate their preservation, retrieval and use are becoming an essential part of many organisations particularly in visual surveillance and medical imaging sectors.

Object oriented techniques have been used successfully in many different applications that range from numerical modelling to web applications. The main benefits, apart from the abstraction power to represent real objects, are the provision for the extensibility needed to create new models and the semantic needed to construct complex objects of similar states (Yourdon, 1994; Bertrand, 1997).

The use of object oriented techniques in information management has been given considerable attention in the past decade (Cattell, 1991; Loomis, 1995). Recent research works have used temporal information and object oriented techniques to explicitly define the relationship between object behaviour over time (Khaddaj, 2004). The ability to examine the continuity of object changes is important for many different applications.

The object oriented approach provides the flexibility to make the changes to attributes and/or behaviour of objects independent of one another, in

order to allow the examination of detailed information of object application model. Therefore, it can be used to identify the pattern of changes within the objects. The simplest way to store changes to objects is that every time a change occurs the whole object is stored again, but this can be prohibitively costly in terms of storage space, and might compromise system performance particularly if objects are updated regularly (fast changes). An alternative is to use object versioning techniques in order to track the evolution of objects. This paper aims to investigate an object model for the management of digital images that will represent the semantics to allow the continuity and pattern of changes of image objects to be determined over time.

In this work we start by considering object orientation's major concepts and information management. Then, object versioning is considered for the development of the proposed model and is used for the determination of the continuous links between different versions of objects and maintaining the metadata of those objects. An object oriented image management model is then considered together with an object-oriented environment for system implementation. Finally, we present some conclusions and suggestions for future works.

## 2 OBJECT ORIENTATION AND INFORMATION MANAGEMENT

The object-oriented approach has the abstraction power to represent real objects. It represents space as a domain populated with independently existing objects that encapsulate attributes and operations. Therefore, this encourages modularity within information systems, while entity relationship models will not show these properties. For example, changes in an object do not necessarily affect the properties of any other object in the system.

The object-oriented approach provides the extensibility needed to create new models through "inheritance" which also promotes hierarchies of objects. It also provides the semantic power needed to construct complex objects of similar states, through "polymorphism", for handling complex attributes and behaviour changes, and the flexibility needed to develop simulation models that can adapt to the changing states of information systems. This approach makes it easier to develop new software from existing ones, thus, promoting reusability.

The object-oriented approach has been used successfully for the unification of temporal and other information related to objects. It is supported by efficient design tools such as UML (Universal Modelling Language), programming tools such as Java, C# and C++, and Object Oriented Database Management Systems (ODBMS) such as Objectivity and Versant. The choice of a particular database, however, clearly depends on the actual application. A relational database is a better choice for a project where relationships among objects are fairly fixed and well known. Object-oriented databases can outperform relational databases at handling complex relationships among objects (Loomis, 1995). The problem becomes acute, however, when the changes are too fast for a database to be redesigned so it can rapidly deliver necessary information.

An object-oriented database could model the presented changes based on a mix of objects and their relationships. For example, if a real life object is represented in object oriented form, rather than as an entry in a database table, associations with other objects (to which it is linked) can automatically "inherit" any changes made, making it easier to track later. At this point it is important to mention that the Enhanced ER Model supports generalisation, aggregation and composition. Moreover, many object oriented features are provided by object-relational database management systems

(ORDBMS) and are supported by SQL3 standard. However, an ORDBMS does not represent a true object oriented database, since it still represents a data-centric system as a relational database.

The ORDBMSs, which have now been supported by most vendors, are much larger and have huge entrenched marketing infrastructure. By comparison, the ODBMS vendors are much smaller. It is clear that in today's complex, rapidly changing world, ODBMSs provide the more flexible, extensible alternative for companies that must act quickly to match the capabilities of their information systems with the needs of their organizations. Users will make choices of database vendors based on many criteria, some of which are addressed here.

## 3 OBJECT VERSIONING

Associating additional temporal information with individual objects provides a means of recording object histories, and thereby allowing the histories of objects and the types of objects to be easily traced and compared. This means that the temporal aspects can also be described by their temporal topological relationships. The object-oriented approach has been used in different ways to effectively track versions of the original object and these include the use of version management (Wachowicz, 1994) and the identity-based method (Hornsby, 2000). Although in this work we are concerned with object versioning, other strategies such as schema versioning can also be considered (Grandi, 2002).

There are a number of methods for dealing with object versioning (Khaddaj, 2004). The first technique stores the complete objects, and any of the versions can be accessed simply by a reference to the particular object. The second approach, which is a relative technique, stores one version as a complete object and the rest of the versions are presented as differences between the current version and the previous version. The method of storing the versions as complete objects is relatively easy to implement in existing database systems. But, this method introduces problems, such as waste of storage space as the number of versions increases. The technique of storing only one complete version and the rest as differences between the current and the previous version is difficult to implement but is suitable for representing continuous and dynamic changes; and it solves the storage space problem of the previous approach. These two approaches have been examined for relational databases (Dadam, 1984).

Using the second approach, changes of objects are handled using version management, starting with a generic object; then first and subsequent changes can be represented as versions. Each version of the object reflects changes of attributes and/or behaviour. Subsequent changes of the versions will generate related dynamic attributes and temporal links to be updated to respective versions. Version management reduces the need for large storage space, since only the generic object or the current object holds the complete attributes and behaviour of the object.

### 3.1 Complete Versions

The first approach can be stated using the following equation (1):

$$Versions(x) = (CV_x(n), CV_x(n-1), \dots, CV_x(n_0)) \quad (1)$$

In equation (1),  $CV_x(n)$  represents the complete version,  $n$  indicates the number of the version,  $x$  is the object and  $n_0$  is the oldest version number. Each version can be accessed directly by reference to the number of the version,  $n$ . Although access to any version is supported directly and all versions have similar access time, storage space can be costly.

### 3.2 Linear Versioning

Using this technique one version is stored as a complete object, and the rest of the versions are presented as differences between the versions (Khaddaj, 2004). The relationship using this approach is based on one-to-one versioning of objects, which means any parent or base object will have only one child or derived object. The technique can be classified into two versioning strategies. The first strategy allows the current version to be calculated from the previous version and is referred to as forward oriented versioning. The second strategy enables the previous version to be evaluated from the current version and is referred to as backward oriented versioning. Using forward linear versioning the temporal relationships between the generic object and versions is given by:

$$Versions(x) = (\Delta_x(n, n-1), \Delta_x(n-1, n-2), \dots, \Delta_x(n_0+1, n_0), CV_x(n_0)) \quad (2)$$

Where  $CV_x(n_0)$  indicates the generic version, which holds the complete attributes and behaviour.  $\Delta_x(k, k')$  represents the difference between the current version ( $k$ ) and the previous version ( $k'$ ) of object  $x$ . As shown in equation (2) access to the current version  $n$  requires  $n-1$  iterations, which

means that this strategy provides faster access time for the oldest version.

In backward linear versioning the current object holds the complete attributes and behaviour. The temporal relationships between the current object and versions is given by:

$$Versions(x) = (CV_x(n), \Delta_x(n, n-1), \Delta_x(n-1, n-2), \dots, \Delta_x(n_0+1, n_0)) \quad (3)$$

As shown in the equation (3), the rest of the versions are expressed as delta to the successor-in-time version, which means that this strategy provides faster access time for the newest versions. As a result, this strategy is bound to be more useful than the previous one for most applications.

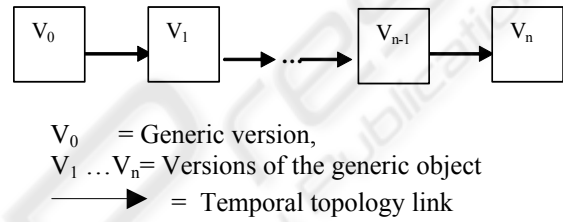


Figure 1: Object versioning.

### 3.3 Branching

This technique is also classified into two versioning strategies. The branch forward oriented strategy is based on one-to-many object versioning (object splitting), which means any parent or base object will have many children or derived objects. The branch backward oriented strategy is based on many-to-one object versioning (object merging) which means any child or derived object will have many parents or base objects. The relationship using the first strategy provides the same access time for all versions:

$$Versions(x) = (\Delta_x(n, n_0), \Delta_x(n-1, n_0), \dots, \Delta_x(n_0+1, n_0), CV_x(n_0)) \quad (4)$$

The relationship using the second strategy provides a faster access time for the current version:

$$Versions(x) = (CV_x(n), \Delta_x(n, n-1), \Delta_x(n, n-2), \dots, \Delta_x(n, n_0)) \quad (5)$$

However, due to the relationships between the current version and the previous ones the values of the versions are re-calculated whenever a new version is created.

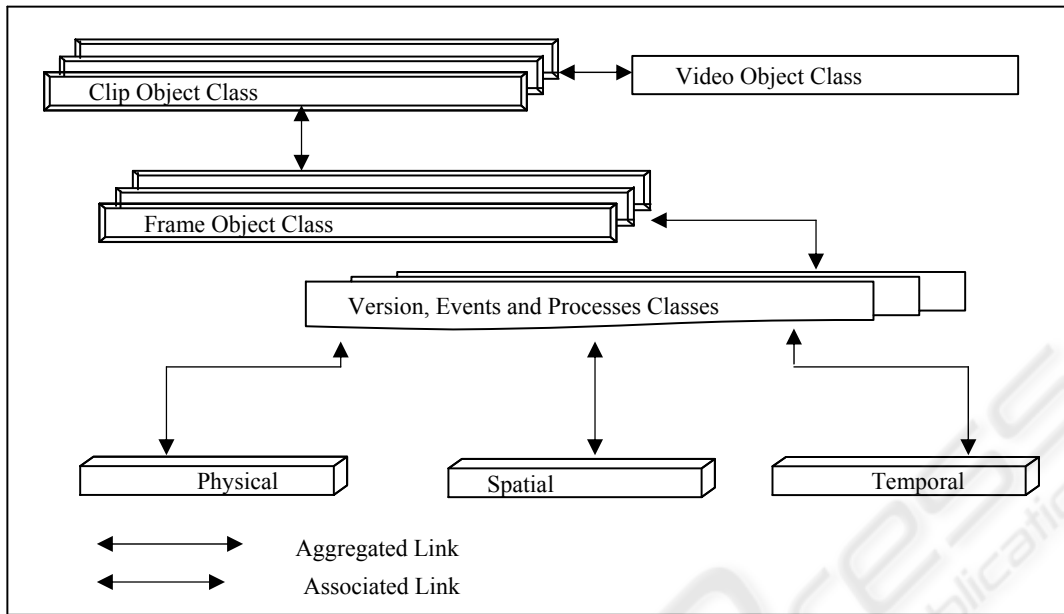


Figure 2: Composite classes of a video object.

## 4 MANAGEMENT OF DIGITAL IMAGES

Video images can now be generated and distributed easily, but what is still needed is support in managing the information contained in those images. This is vital because by putting pieces of information from different images together, the user can generate new knowledge. The ability to efficiently collect, store, manage, analyse, and retrieve digital images remains a major issue.

### 4.1 Object Oriented Model for the Management of Digital Images

Although there has been some work on using object-oriented databases for storing and retrieving images (Dönderler, 2003; William, 2001) not much investigation has been done using the object oriented approach in conjunction with the concept of object versioning. To show the potential benefit of the approach, it is sufficient to consider a typical CCTV monitoring a car park where the background of the scene does not change and the background object can be stored in the database as a generic object, any changes to the scene, for instance a car entering a car park, can be stored in the database as versions instead of storing the whole object, background and changes, again (for simplicity at this point we are assuming stable weather conditions).

Thus, in this work we are concerned with a generic digital image management, context independent, object model reflecting the structure and semantic linking for different types of images. The model should take into account issues like image versions, notification and propagation of changes. Of particular interest in this approach, are scenarios when images are changing fast, and new versions are created whether there is a need for time stamping or not. Clearly, it is more useful when time stamping is required, i.e. where there is a need to keep a history of activities.

The development of a working system, using this model, requires an intelligent video database that provides automatic method of indexing and content-based retrieving of images or video shots generated by video-cameras based on the time they have been recorded and the analysis of their features and content. This can be used to provide an intelligent database solution to the continuous logging and annotation of events in public spaces and environments. To achieve this we propose a new object oriented video model. According to the proposed model a video consists of a number of shots. A shot is consecutive sequence of frames, which are the smallest unit of video data. A frame consists of a number of physical objects, static like building, car park etc. and dynamic objects such as persons, cars etc. Considered classes include the events, processes and the versions class which is fundamental to the system (figure 2).



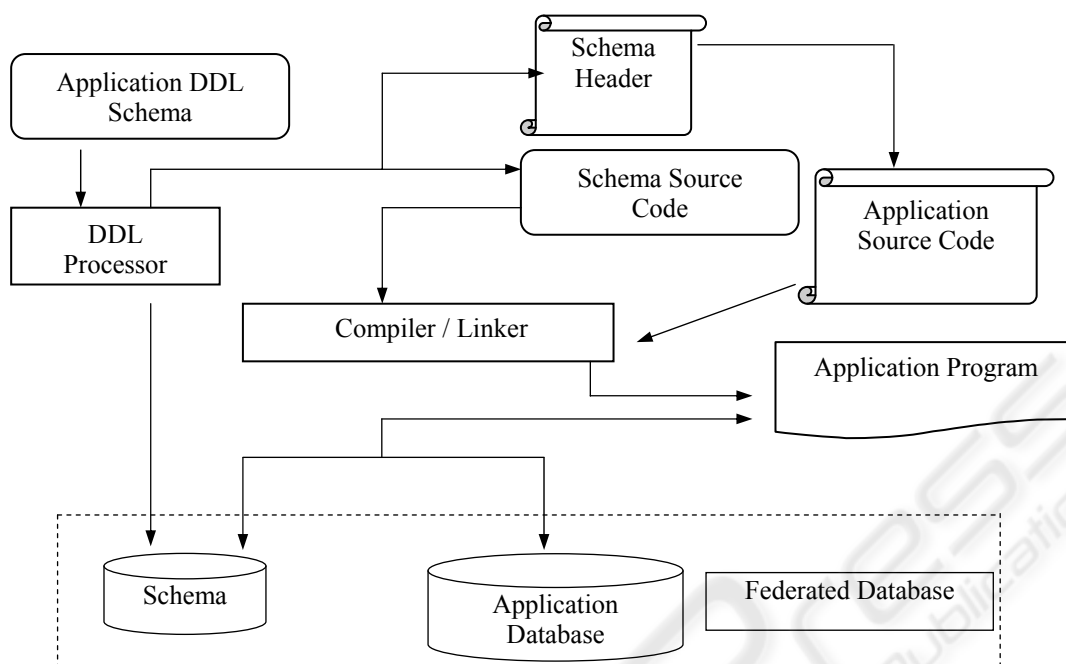


Figure 3: General architecture of the System implementation.

Using the versioning approach a frame object is represented as a generic object, the first object and any subsequent changes can be represented as versions. Each version of the object consists of changes (involving an attribute or behaviour) of the aggregated and the associated classes. Subsequent changes of attributes of the versions will generate related dynamic attributes and temporal links to be updated to the respective versions. The relationships between the generic object and the versions of the object are represented by temporal version management approach (Khaddaj, 2004). To avoid the use of large storage space, only the generic object or the current object holds the complete attributes and behaviour of the object while the other versions represent the changes of their attributes and behaviour. While MPEG-7 defines a large set of description classes for multimedia content-encoding schemes, it is considered complementary to, rather than competing with, data models such as the one proposed in this project.

Moreover, the query technique of the system will be classified into different parts in order to identify the attribute and behaviour changes to the classes and objects of the model. This will provide support of different types of queries such as spatio-temporal, semantic and low level features (color, shape etc.) queries on video data.

## 4.2 System Implementation

A successful implementation of the model will require an Object Oriented Programming Environment (OOPE) and an Object Oriented Database System (OODBS). This approach eliminates the need for mapping the model to an OODBS, since the class structure used in the model, the OOPE and OODBS are consistent. The OODBS considered in this work is based on Objectivity/DB (Objectivity, 2000). The classes are defined in the application schema file, called Data Definition Language (DDL). The DDL processor generates the schema header file and the schema source code which are linked with the application source code. In the application DDL and application source code files, all the classes have their own representation.

Objectivity/DB has the capabilities to represent the various versioning approaches: linear, splitting and merging. Image objects persist by storing the object within the container of the database. Persistent objects are identified using the object identifier (OID) which remains unique within a federated database. Objectivity/DB uses an object handling class to access persistent objects automatically by the DDL process for every persistence class found in the schema header. All the objects can be determined by scanning through the database using iterative scanning functions.

Aggregated relationships between the classes are established in the application source code. Moreover, in order to determine and analyse dynamic changes, the model establishes a temporal relationship between the versions, the event and the manager classes. A dynamic function handles the temporal relationships between the versions, the event and the manager classes.

Moreover, the relationships between the versions allow forward and backward movement. The previous version and the next version to the current version can be obtained by iteration using either backward or forward movement functions. In order to avoid the use of large storage space, only the generic object or the current object holds the complete attributes and behaviour of the object while the other versions represents the changes of their attributes and behaviour.

## 5 CONCLUSIONS

The applications of object oriented techniques to the management of digital images have been discussed in this paper. Particular attention was paid to the concept of object versioning and its applications. The presented object oriented approach provides an integrated framework for effective tracking of the evolution of image objects. It also promotes good temporal modelling, because the temporal attributes and behaviour of the versions are independent but have relationships that enable the tracking of patterns of change. Also, less data storage is required since only the generic object and its versions are stored.

## REFERENCES

- Bertrand M., "Object-Oriented Software Construction", Prentice Hall Publishing International Series in Computer Science. 1997.
- Cattell R.G.G., "Object Data Management, Object-Oriented and Extended Relational Database Systems", Addison-Wesley Publishing, 1991.
- Dadam, P., Lum, V., Werner, H. D., "Integrating of time versions into relational database systems", *Very Large Database Conference*, pp. 509-522, 1984.
- Dönderler, M, Şaykol, E., Ulusoy, Ö., Güdükbay, U., "BilVideo: A Video Database Management System", *IEEE Multimedia*, Vol. 10, No. 1, pp. 66-70, 2003.
- Grandi F., Mandreoli F., "A Formal Model for Temporal Schema Versioning in Object-Oriented Databases". A Timecenter Technical Report, 2002.
- Hornsby K., Egenhofer M., "Identity-Based Change: A Foundation For Spatio- Temporal Knowledge Representation", *International Journal of Geo Information Systems* 14(3), pp 207-224, 2000.
- Khaddaj S., Adamu A., M. Morad, "Object versioning and Information Management", *Journal of Information and Software Technology*, 46(y) Elsevier Science, June, pp. 491-498, 2004.
- Loomis M.E.S., "Object Databases; The Essentials", Addison-Wesley Publishing, 1995.
- Objectivity/DB, "Complete handbook for objectivity/C++ Instruction Manual", 2000.
- Wachowicz, M. and Healey, R. "Towards temporality in GIS" *Innovation in GIS I*, by Worboys M. F. Vol 1 pp.105- 115, Taylor & Francis, 1994.
- William I. Grosky and Peter L. Stanchev, "Object-Oriented Image Database Model", 16th International Conference on Computers and Their Applications, pp 93-98 2001.
- Yourdon, E. "Object-Oriented System Design: An Integrated Approach", Yourdon Press, 1994.