Journal of Computer Science 6 (1): 47-51, 2010
ISSN 1549-3636

An Integrated Model for the Versioning of Image Objects

S. Khaddaj
Faculty of Computing, Information System and Mathematics,
Kingston University, Kingston upon Thames KT1 2EE, UK

Abstract: Problem statement: The efficient management and retrieval of the enormous volumes of information generated by many imaging applications is still a challenging problem. **Approach:** In this study we are particularly interested with the use of object oriented techniques for the management of digital images. Indeed, it is widely accepted that the use of data abstraction in object oriented modeling enables real world objects to be well represented in information systems. **Results:** Object orientation is well suited for digital images management systems, which require the ability to handle multiple type content. This study proposes a conceptual model, based on object versioning techniques, which represents the semantics in order to allow the continuity and pattern of changes of images to be determined over time. **Conclusion/Recommendations:** By first storing a generic image object and then storing any changes to the object as versions, a major reduction in storage can be achieved.

Key words: Object oriented modeling, object versioning, image management systems

INTRODUCTION

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

Object oriented techniques have been used successfully in many different applications that range from numerical modeling 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, 1993; Bertrand, 1993).

The use of object oriented techniques in information management has been given considerable attention in the past decade (McHaney, 2003; Chaudhri and Loomis, 1997). Recent research studies have used temporal information and object oriented techniques to explicitly define the relationship between object behavior over time (Khaddaj, 2004). The ability to examine the continuity of object changes over time is very important for many different applications.

The object oriented approach provides the flexibility to make the changes to attributes and/or behavior 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 study 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 study 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 study.

MATERIALS AND METHODS

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 behavior 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 Universal Modeling Language (UML), 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 (Chaudhri and Loomis, 1997). 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 generalization, 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.

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 (Fig. 1). 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).

There are a number of methods for dealing with object versioning (Khaddaj, 2004). The first technique stores the versions as 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 behavior. Subsequent changes of the versions will generate related dynamic attributes and temporal links to be updated to respective versions. Version management reduces the

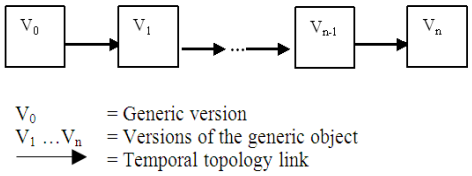

Fig. 1: Object versioning

need for large storage space, since only the generic object or the current object holds the complete attributes and behavior of the object.

Complete versions: The first approach can be stated using the following Eq. 1:

$$\text{Versions}(x) = (CV_x(n), CV_x(n-1),\ldots..,CV_x(n_0)) \quad (1)$$

In Eq. 1:
$CV_x(n)$ = Represents the complete version
n = Indicates the number of the version
x = The object
n_0 = 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.

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:

$$\text{Versions}(x) = (\Delta_x (n, n-1), \Delta_x (n-1,n-2),..., \Delta_x(n_0+1, n_0), CV_x(n_0) \quad (2)$$

Where:
$CV_x(n_0)$ = Indicates the generic version, which holds the complete attributes and behavior
$\Delta_x (k, k')$ = Represents the difference between the current version (k) and the previous version (k') of object x

As shown in Eq. 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 behavior. The temporal relationships between the current object and versions is given by:

$$\text{Versions}(x) = (CV_x(n), \Delta_x(n,n-1), \Delta_x (n-1,n-2),.., \Delta_x(n_0+1, n_0) \quad (3)$$

As shown in the Eq. 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.

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:

$$\text{Versions}(x) = (\Delta_x (n,n_0), \Delta_x (n-1,n_0),...., \Delta_x(n_0+1, n_0), CV_x(n_0) \quad (4)$$

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

$$\text{Versions}(x) = (CV_x(n), \Delta_x (n,n-1), \Delta_x (n,n-2),.., \Delta_x(n, n_0) \quad (5)$$

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.

RESULTS

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, analyze and retrieve digital images remains a major issue.

Object oriented model for the management of digital images: Although there has been some study on using object-oriented databases for storing and retrieving images (Dönderler, 2005; Grosky and Stanchev, 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). The approach, as mentioned earlier, will reduce dramatically the size of the stored information.

Thus, in this study 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 and dynamic objects such as persons, cars etc. Considered classes include the events, processes and the versions class which is fundamental to the system (Fig. 2).

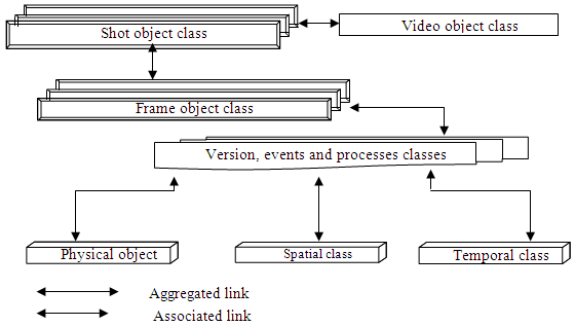

Fig. 2: Composite classes of a video object

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

System implementation: A successful implementation of the model will require an Object Oriented Programming Environment (OOPE) and an Object Oriented Database System (OODBS) (Fig. 3). 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 study is based on Objectivity/DB (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 and versions in the database 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,

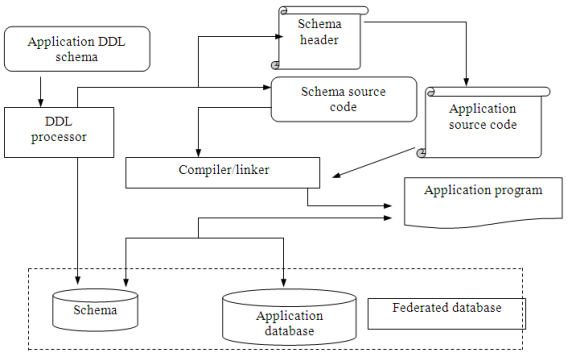

Fig. 3: General architecture of the system implementation

in order to determine and analyze 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.

The approach has proved to be effective in reducing the size of stored images. It can be employed in many applications for example in car park administration where video surveillance and monitoring are commonly used with the need of continuous logging and annotation of events. The gain in storage depends on how busy the car park is, in peak hours up to 30%, during the day around 50% and very late evening/night around 70%. The gain is achieved by separating the background of the scene, which does not change (assuming stable weather conditions) and any changes to the scene. The background object can be stored in the database as a generic object and any changes to the scene can be stored in the database as versions.

DISCUSSION

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 behavior) 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 discussed earlier. To avoid the use of large storage space, only the generic object or the current object holds the complete attributes and behavior of the object whiles the other versions represents the changes of their attributes and behavior. 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 research.

CONCLUSION

The applications of object oriented techniques to the management of digital images have been discussed in this study. 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 modeling, because the temporal attributes and behavior 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., 1993. Systematic concurrent object-oriented programming. Commun. ACM., 36: 56-80. DOI: 10.1145/162685.162705

McHaney R. and C Hagmann, 2003. Object-oriented database. Tailor Francis Group, 20: 75-83. DOI: 10.1081/E-ELIS-120008836

Dadam, P., V. Lum and H.D. Werner, 1984. Integrating of time versions into relational database systems. Proceeding of the Conference on Very Large Database, pp: 509-522. DOI: 10.1234/12345678

Dönderler, M.E., E. Şaykol, Ö. Ulusoy and U. Güdükbay, 2005. BilVideo: Design and implementation of a video database management system. Multimedia Tools Appli., 27: 79-104. DOI: 10.1007/s11042-005-2715-7

Grosky, W.I. and P.L. Stanchev, 2001. Object-oriented image database model. Proceeding of the 16th International Conference on Computers and their Applications, Mar. 28-30, Seattle, Washington, pp: 94-97.

Hornsby, K. and M. Egenhofer, 2000. Identity-based change: A foundation for spatio-temporal knowledge representation. Int. J. Geo Inform. Syst., 14: 207-224. DOI: 10.1145/956676.956688

Khaddaj, S., A. Adamu and M. Morad, 2004. Object versioning and information management. J. Inform. Software Technol., 46: 491-498. DOI: 10.1016/j.infsof.2003.10.002

Chaudhri, A. and M.E.S Loomis, 1997. Object Databases in Practice. Prentice Hall PTR, ISBN: 13: 978-0138997250, pp: 336.

Objectivity, 2000. Complete Handbook for Objectivity/C++ Instruction Manual. http://www.objectivity.com/

Wachowicz, M. and R. Healey, 1994. Towards Temporality in GIS. In: Innovation in GIS, Worboys, M.F. (Ed.). Taylor and Francis, ISBN: 3540411771, pp: 105-115.

Yourdon, E., 1993. Object-Oriented System Design: An Integrated Approach. Prentice Hall, ISBN: 0136363253.