AN OBJECT ORIENTED APPROACH FOR DOCUMENT MANAGEMENT

S. Khaddaj
School of Computing and Information Systems, Kingston University, Kingston upon Thames KT1 2EE, UK.

Keywords: Object Oriented Modelling, Object Versioning, Document Management Systems

Abstract: 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 document management. Object orientation is well suited for such systems, which require the ability to handle multiple types content. However, the matter of how to deal with the reuse and management of existing documents over time remains a major issue. This paper aims to investigate a conceptual model, based on object versioning techniques, that will represent the semantics in order to allow the continuity and pattern of changes of documents to be determined over time.

1 INTRODUCTION

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, Martin and Odell 1995, Bertrand 1997].

The use of object oriented techniques in information management has been given considerable attention in the past decade [Cattell 1991, Won 1998, Loomis 1995]. Recent research works have used temporal information and object oriented techniques to explicitly define the relationship between object behaviour over time [Khaddaj et al.]. 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 behaviour of objects independent of one another, in order to allow the examination of detailed information of object application models. 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 a document management object model that will represent the semantics to allow the continuity and pattern of changes of objects to be determined over time.

2 OBJECT ORIENTATION AND INFORMATION MANAGEMENT

The object-oriented approach has the abstraction power to represent real objects and it 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 C++ and Java and, more recently, by object oriented databases 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.

Khaddaj S. (2004).
AN OBJECT ORIENTED APPROACH FOR DOCUMENT MANAGEMENT.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 558-561
DOI: 10.5220/0002620705580561

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 and Healey 1994] and the identity-based method [Hornsby 2000]. Moreover, other strategies such as schema versioning can also be considered [Grandi 2002].

There are a number of methods for dealing with object versioning. The first technique stores the versions as complete objects, which is easy to implement in existing database systems but it is costly in storage space. The second approach 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 technique is difficult to implement but it solves the storage space problems. These two approaches have been examined for relational databases [Dadam et al., 1984].

3.1 Complete Versions

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

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

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.

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. 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 [Dadam et al., 1984]. Using forward linear versioning the temporal relationships between the generic object and versions is given by:

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

Where 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:

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

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.

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:

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

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

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

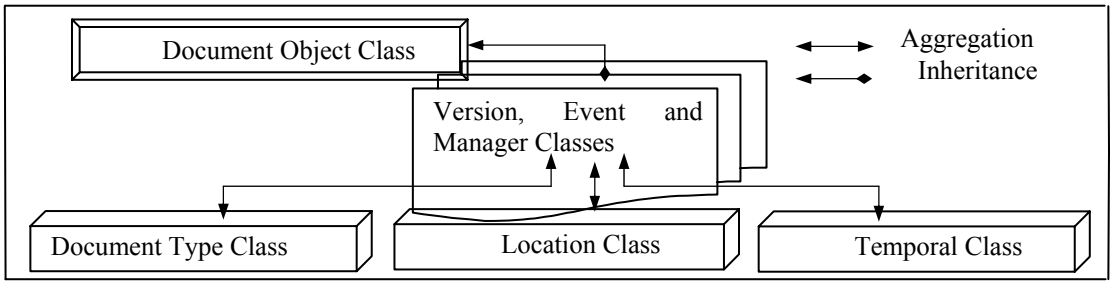

Figure 1: Composite classes of a document object

4 DOCUMENT MANAGEMENT

Documents can now be generated and distributed easily, but what is still needed is support in managing the information contained in those documents [Barth 2000, Bielawski 1998]. This is vital because by putting pieces of information from different documents together, the user can generate new knowledge [Davenport et al., 1998]. The ability to collect, store, manage, analyse, retrieve and utilise information about documents, and present information in text, graphics and, increasingly, multimedia form has received considerable attention in the past few years [Bielawski and Boyle 1998, Outsell 2001, Outsell 2001]. However, the matter of how to deal with the reuse and management of existing information over time remains a major issue.

4.1 Object Oriented Model for Document Management

In this section we are concerned with a generic document management, context independent, object model reflecting the structure and semantic linking for different types of documents. The model should take into account document structuring and content referencing; and includes issues like document versions, ownership, notification and propagation of changes, particularly in a collaborative environment.

Of particular interest in this approach, are scenarios when documents are regularly updated, 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 and changes in managed documents, such as user manuals, online help, tourist guides, web applications, etc. The idea is to simplify the management of all types of documents such as scanned paper documents, faxes, emails, word processing generated reports, spreadsheets, html forms, and so on.

Using this approach, changes to documents are handled by version management. A version of the object consists of composite classes as in figure 1. The aggregated composite classes include a document type class (documents can be classified according to their types which can take the form of texts, graphics, multimedia etc, and can be regarded as derived classes from the type base class), location class and temporal class. The associated composite classes include manager class and event class. The location class deals with queries about the location of the object document within the federated database (including the ability to search documents by either context or index term, text extraction and full text search engine). The type class deals with queries about the features of an object (e.g. length, content types etc). The temporal class deals with the queries about the time attributes of the object (e.g. when was the document created). Furthermore, an event class deals with the changes (and their causes) of the document object (e.g. adding new manuals after an operating system upgrade). And, a manager class with persistent object store, including the ability to store documents, to control the access to documents, to deal with the effect of the changes of the object, to assure changes are not confused, and to co-ordinate documents transformation, extension, etc.

4.2 The Version Class

As figure 2 illustrates, a document object is represented as a generic object and any subsequent changes are represented as versions. Each version of the object consists of changes (involving an attribute or behaviour) of the aggregated classes (type, location and temporal) and the associated class (event and manager). 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 [Dadam 1984]. To save storage space, only the generic version hold the complete attributes and behaviour of the object.

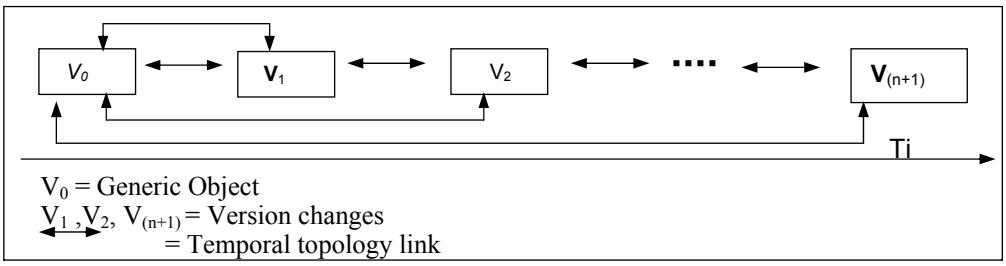

Figure 2: Relationship between the versions and generic object

4.3 System Implementation

A successful implementation of the model will require an Object Oriented Database System (OODBS). The OODBS considered in this work is based on Objectivity/DB [Objectivity/DB 2000]. The classes (version, temporal, location, type, event and manager) 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.

Objectivity/DB has the capabilities to represent the various versioning approaches: linear, splitting and merging. As discussed earlier, simple changes are represented by a linear versioning method while complex changes, involving splitting and merging, are represented by branching. document 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.

5 CONCLUSIONS

The applications of object oriented techniques to document management 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 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

Yourdon, E. “Object-Oriented System Design: An Integrated Approach”, Yourdon Press, 1994.

Martin J., Odell J.J., “Object-Oriented Methods: A Foundation”, Prentice Hall, Englewood Cliffs, 1995.

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.

Won K, Lochovsky, F., “Object-Oriented Concepts, Databases, Applications**”,** ACM press Frontier Series. Addison-Wesley Publishing, 1989.

Loomis M.E.S., “Object Databases; The Essentials”, Addison-Wesley Publishing,1995.

Khaddaj S., Adamu A., M. Morad, "Object versioning and Information Management”, Journal of Information and Software Technology,to appear.

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.

Dadam, P., Lum, V., Werner, H. D., “Integrating of time versions into relational database systems”, *Very Large Database Conference*, pp. 509-522, 1984.

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.

Grandi F., Mandreoli F., “A Formal Model for Temporal Schema Versioning in Object-Oriented Databases”. A Timecenter Technical Report, 2002.

Barth S, “KM Horror stories”, Knowledge Management 3, No 10, pp 36-40, 2000.

Bielawski L., Boyle J, “Electronic Document Management Systems: a User Centered Approach for Creating, Distributing and Managing Online Publications”, Upper Saddle River, P Hall 1998.

Davenport T. H, De Long D. W. and Beers M. C., “Successful Knowledge Management Projects”, Sloan Management Review, 39, pp 43-57, 1998.

Outsell Inc. “ Taxonomies: Structuring Today’s Knowledge Systems”, Infor Briefing 4, pp 1-18,2001.

Outsell Inc., “ Knowledge Management: It’s All About Behavior”, InforBriefing 4, pp 1-16, 2001.

Objectivity/DB, “Complete handbook for objectivity/C++ Instruction Manual”, 2000.