Asian Journal of Information Technology 4 (9): 845-851, 2005
© Grace Publications, 2005

An Object Oriented Framework for the Management of Compound Documents

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

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.

Key words: Object oriented modelling, object versioning, document management systems

INTRODUCTION

Electronic document management systems, in which documents are collected, organised and categorised to facilitate their preservation, retrieval and use have become an essential part of many organisations in order to retain their business competitive edge. Historically, relational databases have been the most popular for the implementation of such systems however, recently object oriented concepts and databases have been considered for the management of complex compound documents.

Object oriented concepts 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[1,2]. The attributes and behaviour are encapsulated within the objects; therefore a network of relationships among objects can be easily established[3] .

The use of object oriented techniques in information management has been given considerable attention in the past decade[4-6]. Recent research works have used temporal information and object oriented techniques to explicitly define the relationship between object behaviour over time[7]. 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. In this study, object tracking and evolution include not only attributes changes to homogenous objects, but also major changes that lead to transforming/destroying existing objects and creating new ones, for example using object splitting and merging.

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

Corresponding Author: Souheil Khaddaj, School of Computing and Information Systems, Kingston University, Kingston upon Thames, Surrey KT12EE, UK Tel: +44 20 8547 2000 Fax: +44 20 8547 7972

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 C++ and Java and, more recently by 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[6]. 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.

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[8] and the identity-based method[9]. Although in this work we are mainly concerned with object versioning, other versioning strategies such as schema versioning can also be considered[10].

There are a number of methods for dealing with object versioning. 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[11].

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.

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

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

In Eq. 1, CV 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 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. There are many different strategies for working out the differences, which are only restricted by storage space and performance. 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[11]. Using forward linear versioning the temporal relationships between the generic object and versions is given by:

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

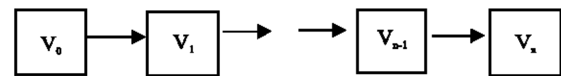
Where $CV_x(n_0)$ is the complete version of object x whiles n_0 indicates the generic version, which holds the complete attributes and behaviour. Δ_x (k, k') represents the difference between the current version (k) and the previous version (k') of object x. As shown in Eq. 2 access to the current version n requires $n-1$ iterations, which means evaluating delta version Δ_x (n_{0+1}, n_0) followed by delta version Δ_x (n_{0+2}, n_{0+1}), then the next version up to delta version Δ_x (n, n-1). This forward oriented versioning strategy equation provides faster access time for the oldest version (Fig. 1).

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:

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

As shown in the Eq. 3, the rest of the versions, apart from the current version, 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: In this technique, one version is stored as a complete object and the rest of the versions are presented as differences between that version and other versions. There are several different strategies for working out the differences, which are again only restricted by storage space and performance. 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 (Fig. 2). 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 (Fig. 2).



V_0 = Generic Version, V_1 ... Vn = versions of generic object
→ = Temporal toplogy link

Fig. 1 Linear versioning

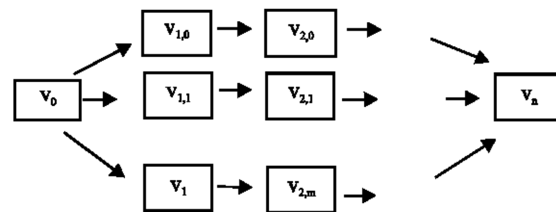


Fig. 2 : Combined strategies

V_0 = Generic version (Parent), V_n = Object merging
V_{n0}...Vn,m = Object splitting , $V_{1,0}$... V_{i-n} Linear Versioning
Temporal Topology link

The relationship using branch forward versioning strategy provides the same access time for all versions:

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

In Eq. 4, as before, Δ_x is the delta version represents the difference between the previous version and the current version. In Eq. 4, version number n_0 represents the generic version of an object x and this provides equal access time for all the versions. The value of the delta remains unchanged when versions are created. All versions are derived only from the generic one.

The relationship using branch backward versioning is based on many-to-one versioning of objects and is given by:

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

This strategy provides a faster access time for the current version. 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.

Combined strategies: Different versioning strategies might be more suitable for different applications. Linear versioning can be applied for attributes changes and/or behaviour changes, while branch versioning is required for splitting and merging objects. In many applications however, it might be necessary to apply both strategies (Fig. 2). As shown in Fig. 2, a generic object (V_0) is split into a number of objects ($V_{1,0}, V_{1,1} \ldots V_{1,m}$) each of which then follows its own linear transformation ($V_{1,0}, V_{2,0} \ldots V_{n-1,0}$), and eventually the objects are merged to form one object (V_n).

DOCUMENT MANAGEMENT

Most events and processes in an organisation are initiated, accompanied or formalised by some form of documentation. Documents can now be generated and distributed easily, but what is still needed is support in managing the information contained in those documents[12,13]. This is vital because by putting pieces of information from different documents together, the user can generate new knowledge[14]. 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[13,15,16]. However, the matter of how to deal with the reuse and management of existing information over time remains a major issue.

Document management systems: There are a number of elements of a Document Management System which include the software to manage documents across an organisation at the core of which is the database used for storage, retrieval, etc. of documents. These also include workflow management systems and more recently knowledge management. Other elements will include infrastructure, authoring tools, distribution etc. However, in this work we are mainly concerned with information database storage and retrieval. Traditionally relational databases have been the most popular, but recently many Document Management Systems are moving toward Object-oriented Database Management Systems (ODBMS).

Historically, document management systems have aimed to deal with two main types of documents; static unflexible documents which are produced by scanners and other devices and editable changing documents, which are produced by many software packages such as word processing and spreadsheets etc. Systems supporting static documents focus on access, with input, indexing and retrieval, while systems supporting editable documents focus on creation, authoring, workflow etc. However, almost all documents have some structure. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote). Mechanisms to identify structures in a document are provided by markup languages such as XML which defines a standard way to add markup to documents.

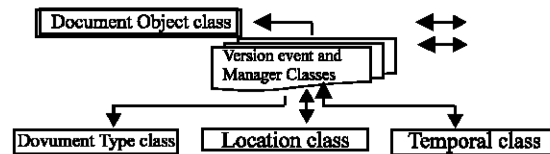
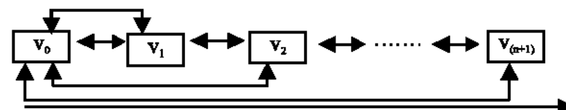


Fig.3: Composite classes of a document object



V_0 = Generic Object
V_1, $V_{2\ V(n+1)}$= Version changes
↔ = Temporal topology link

Fig. 4: Relationship between the versions and generic object

There have been a massive increase in the number of applications currently being developed that or make use of XML documents. However, such documents refers not only to traditional documents, but also to the miriad of other XML data formats which include vector graphics, mathematical equations, object meta-data etc.

However, there are a number of approaches to store and query XML data. Storing XML data in a file system is straightforward but does not support querying XML data. Other generic approaches that store documents without any user interaction, and which provide for the storage and retrieval of different types of XML documents, e.g. XSL documents etc., using the same relational schema for storing. However different strategies to completely decompose arbitrary XML documents into relational tables are required[17]. Other methods to store XML documents in relational or object-relational databases that is based on an adaptable fragmentation[18].

As a result of the massive increase in the amount and complexity of the generated documents there has been a major need to move away from the management of static/editable documents toward complex, compound documents which are constantly changing. These are not usually tied to a particular application or software and they include information about their content and structure. In this way, documents are reflecting the trend toward object-oriented architectures, where information is contained in objects which can be considered as units of information of a finer granularity than traditional documents, and which also contain information about themselves and their originating applications. Documents using this approach can use many object oriented concepts such as aggregation and containment where new specific documents can be created from existing objects which can also be reused. Documents objects will have attribute such as author, dates, status etc, and behaviour such as access control, workflow processing etc. Using association links to other external objects representing images, datasets etc. can be established.

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. The model makes use of the techniques discussed earlier, particularly versioning and will tackle problems related to document storage and tracking.

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 Fig. 3. 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.

The version class: As Fig. 4 shows, 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[11]. 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 versions represent the

intermediate changes of the attributes and behaviour. The temporal relationships between the current object and versions can be stated by either Eq. 2 or Eq. 3. According to this approach, when a document object splits, the generated dynamic attribute locates the versions and creates temporal links between the previous version and the new versions. Similarly, when document objects merge, the generated attributes will establish the location of the new version and create temporal links between the previous and the new version.

Thus, a version of an object consists of changes (attribute or behaviour) of the type, location and temporal classes. Subsequent attributes and behaviour of the classes are automatically updated to the respective class. Each attribute or behaviour change is contained in a version, linked bi-directionally to the respective type, location and temporal classes. Also, the attribute and behaviour changes of the versions of the object are linked respectively to the previous and next changes.

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[19]. 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. In the application DDL and application source code files, all the classes have their own representation (Fig. 5).

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

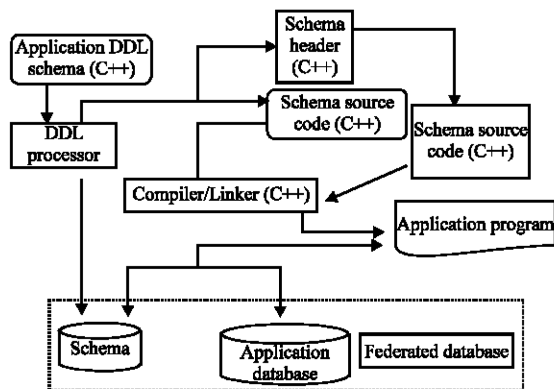


Fig. 5: General architecture of the system implementation

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.

Aggregated relationships between the version class and the type class, the location class and the temporal class 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.

As indicated earlier 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.

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 the changes to the object, which are represented as versions are stored.

Finally, although the author has focused on a particular application, the ideas discussed in this paper can be easily applied to other systems and applications. Further comprehensive testing and evaluation of the approach and its implementation will be carried out and reported in future study, particularly with large and complex data.

REFERENCES

1. Yourdon E., 1994. Object-oriented system design: An Integrated Approach, Yourdon Press.
2. Martin, J. and J.J. Odell, 1995. Object-Oriented Methods: A Foundation, Prentice Hall, Englewood Cliffs, NJ.
3. Bertrand, M., 1997. Object-Oriented Software Construction. Prentice Hall Publishing International Series in Computer Science.
4. Cattell, R.G.G., 1991. Object Data Management, Object-Oriented and Extended Relational Database Systems, Addison-Wesley Publishing.
5. Won, K and F. Lochovsky, 1989. Object-Oriented Concepts, Databases, Applications. ACM press Frontier Series. Addison-Wesley Publishing.
6. Loomis, M.E.S., 1995. Object Databases; The Essentials, Addison-Wesley Publishing.
7. Khaddaj, S., A. Adamu and M. Morad, 2004. Object versioning and Information Manag. J. Inform. and Software Technol., 46: 491-498.
8. Wachowicz, M. and R. Healey, 1994, Towards Temporality in GIS. In: By Worboys M.F. (Ed.). Innovation in GIS I., 1: 105-115.
11. 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.
9. Hornsby, K. and M. Egenhofer, 2000. Identity-based change: A foundation For Spatio-temporal knowledge representation. Intl. J. Geograph. Inform. Systems, pp : 207-224.
11. Grandi, F. and F. Mandreoli, 2002. A formal model for temporal schema versioning in object-oriented databases. A Timecenter Technical Report TR-68.
12. Barth, S., 2000. K.M. Horror stories. Knowledge Management, 3: 36-40.
13. Bielawski, L. and J. Boyle, 1998. Electronic document management systems: A user centered approach for creating, distributing and managing. Online Publications, Upper Saddle River, NJ: Prentice Hall PTR.
14. Davenport, T.H., D.W. De Long and M.C. Beers, 1998. Successful knowledge management projects. Sloan Management Review, 39: 43-57.
15. Outsell, Inc., 2001. Taxonomies: Structuring today's knowledge systems. Information about Information Briefing, 4: 1-18.
16. Outsell, Inc., 2001. Knowledge management: It's all about behavior, Information about Information Briefing, 4: 1-16.
17. Williams, K., 2002. Professional XML Databases. Wrox Press Ltd.
18. Christian, S., 2001. An approach to the model-based fragmentation and relational storage of XML-documents. 13th GI-Workshop Grundlagen von Datenbanken.
19. Objectivity/DB, 2003. Complete handbook for objectivity/C++ Instruction Manual.