On Temporal Versioning in Object-Oriented Databases

Jiang Lü, Peter Barclay and Jessie Kennedy
Department of Computer Studies, Napier University, Edinburgh EH14 1LT, UK.
email: <jiang, pete>@dcs.napier.ac.uk

Abstract This paper describes the development of the data model TVM which contains a new concept, temporal versioning. This generalises the ideas of temporal databases and version management to object-oriented databases. TVM provides a new approach for data modelling and management; it allows the user to define a multi-dimensional temporal storage space and to model the evolution of objects within that space. It supports historical versions and alternative versions of objects at instance level and as well as schema level.

1 Motivation and Objectives

There are two application areas where temporal data [McKenzie 1991] and versions [Sciore 1991] have become particularly important: temporal databases and version control and management systems. Although these two areas have a similar conception, the systems developed for each area have different concerns. Versions and temporal data are different concepts; *temporal data* reflects the states of objects in a more time-oriented way; this feature does not apply to *version*s. Techniques developed in temporal databases cannot solve the problems of versioning. On the other hand, the techniques developed in the area of version control and management lack of the ability of support time-varying data. A temporal database is one in which the information about the states of entities (objects) in the database is a function of time; it is primarily concerned with the power and expressiveness of temporal query languages [Kline 1993]. Version management and control systems support the design of engineering objects. They have focused on understanding the system-level requirements of versioning. In particular, the questions of how to support alternative versions, configurations, long transactions and control access to versions have received substantial attention.

These two areas have solved different but related problems from different angles. However, issues such as how these different solutions can be combined have not yet been tackled. Many applications require databases that support both temporal data and version management (such as design of engineering objects). Figure 1 shows a Car-Manufacturing Database. There are several types of *lock* which can be used in this car model. These locks are an *alternatives* of each other and valid at same time. The historical information of *engine design* is important, the updated data is need to be preserved. It is considered *historical (temporal)* data. For the *safety system*, both of the *temporal and alternative* data are important. It is needed to be maintained at same time. However most of proposed temporal database models support neither alternative versions nor schema versioning and version management do not have time aspects support. The combinations of the techniques developed in these two areas will provide a more general solution to these problems.

The objective of this study is to present a simple and theoretically consistent data model. The principle is to treat all the relevant information in a single data model. We introduce a new data concept “temporal versioning”, which provides a uniform way of understanding temporal data and versions. We present a data model, called temporal versioning model (TVM), which incorporates temporal versioning semantics of the real world into the object-oriented database model. TVM supports multiple dimensional time to overcome the limitation of existing temporal database models which support one valid time attribute and one valid transaction time only [Snodgrass 1994]; a temporal version can be accessed by its time information, or its identifier or values of properties; this aims to overcome the limitations of accessing temporal data only by its time information or by its version identifier. Alternative versions, schema history and other features are also supported by TVM.

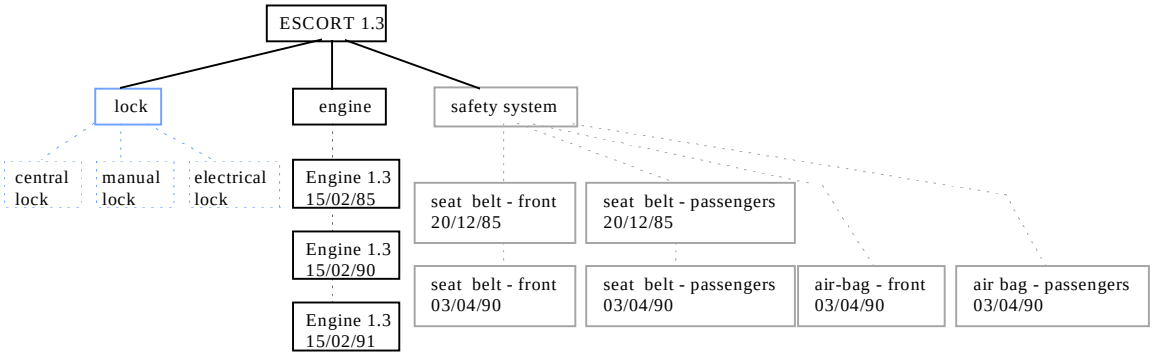

Figure 1

We adopt the framework for TVM by extending the remarks posed in [Soares 1995]. By analysing the results in [Agrawal 1991], we identify the following requirements:

- Maintenance of an enterprise history - a good DBMS will enable the users to maintain and manipulate a history of changes to their data. These histories are for objects and schemas.
- Access to the history of an enterprise by its version values (including its identifier) as well as its time information - It should be possible to access the history of an enterprise by its time information; the property values and other version values could also be used in the query specification. It should be possible to express the temporal order inherent in a temporal object.;
- Current version and temporal version - Temporal is referred as *non-current*, since for each schema or object, it may have many versions at different times.
- Support alternatives - It should allow users to explore several configurations simultaneously. More than one version for an object at a particular time may be allowed.
- Versionable and non-versionable - The value of non-versionable class may be modified without creating a new version of the object. Changes to versionable classes have to be made on a new version of the object.

2 Temporal Versioning Model (TVM)

2.1 Preliminaries

We regard the entire process of evolution of entities over time as happening within a conceptual *temporal storage space*. The evolution of an object within temporal space can be described as a series of versions (or states) of an object. Temporal data may exist in the form of an interval within each time dimension. The assignment of a coordinate system provides a one-to-one mapping between the position of the time-space and the state of an object. e.g. the coordinates of a point may be represented by Cartesian coordinates. Every position within the temporal storage space can be described by a set of corresponding values of each time dimension. A time-interval represents the life-span of the version of an entity. An object-oriented database model NOM (Napier Object Model) is used to represent the states of objects. (NOM is a simple data model intended to allow object modelling of data at a conceptual level. It was first presented in [Barclay 1991] and is described fully in [Barclay 1993]. The data defination and manipulation languages NOODL (Napier Object Oriented Data Language) may be used to specify enterprises modelled using NOM. A NOODL schema contains a list of class definations, which show the name and ancestors of each class. A class definition also contains operations, constraints, and triggers. The GNOME (Generic Napier Object Model Environment) is an implementation of NOM in the persistent programming language Napier88 [Dearle 1989].)

For example, in order to record employee E's salary history, the salary history must be maintained along two time dimensions: *valid time*-denotes the time when a fact becomes effective in reality, *transaction time* - concerns the time of storage of information in the database [Mckenzie 1991]. Figure 2 illustrates our modelling approach. E_1, E_2 and E_3 represent three different states of an object E in temporal space T (valid time, transaction time).

Within t_{ij}, i represents the state sequence order and j represents different time dimensions. Therefore, t_{ij} is the "shadow" of state i of object E at time dimension j. It can also be viewed as three different versions of E. In the beginning E's state is E_1 at the time of t_{1t}, t_{1v}. At the time t_{2t}, t_{2v}, E has an update and its state is

E_2. Finally, at t_{3t}, t_{3v}, E has another update, as the result of the update, the state of E becomes E_3.

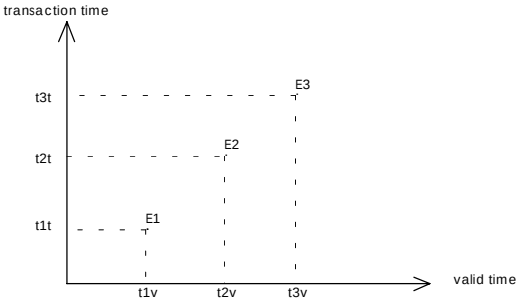

Figure 2

2.2 Basic concepts

The main features of TVM can be summarised as follows:

- Each schema may have many versions. A new version with its own life-span is created after each update.
- Each object (instance of class) may have many versions and each version has its own life-span.
- An instance of a schema and its relevant data construct an *schema-instance-hierarchy*, which also has its life-span.
- Different versions of schema may get accessed by its time information and version-indentifier. Different versions of object may get accessed by its property values, time information and version identifier.

We will continue with an introduction of the main components of TVM, and the description of mechanisms for their support and management. Our discussion will mainly at the conceptual level, but some remarks may address at the implementation level.

There are two types of schema in TVM: *versionable* and *non-versionable schema.* Changes in the versionable schema and its instances may result the creations of new version. Within the versionable schema, there may be two types of classes, *versionable class* and *non-versionable class*. Modifications on non-versionable schemas, or non-versionable classes and their instances will result the previous data being dismissed, because for these data, the temporal versions are assumed not to be important.

schema type		
non-versionable schema	**versionable schema**	
non-versionable classes	*non-versionable classes*	*versionable classes*

For each time dimension, two special time-related properties *FROM* and *TO* are associated with each version. The time of creating a version is recorded in its *FROM*, and its *TO* is assigned as * to indicate that this version is valid until further change. The historical and future time are described by their specific time data (timestamps). A symble ⊗ is used to indicate *the end of future*. Whenever the version has an update, the time of the update is assigned to *TO*. The time data is associated with non-temporal data in two levels: object instance level and schema level in each time dimension, or instance-oriented data and schema-oriented data.

If more than one version of an object is valid at a certain time, these versions may be distinguished by version-indentifiers. For example, in order to distinguish the cars with different colours within one model, such version identifiers are required, these different values are called *alternative*s.

Similar to handling object versions, a version-identifier may be defined by user to distinguish different schema versions which are valid at same time, and a version class can be adopted for keeping schema versions.

TVM can maintain the history of changes to both the schema and its data. An instance of a schema and its data (all its object instances) constructs an *schema-instance-hierarchy* which is a combination of versions of various objects and the version of the schema that was used to create them.

This is different from the mechanisms used in ORION [Chou 1987], in which such as ‘version-counter’ and ‘next-version-number’ are employed to link the different versions of same versionable object. In TVM, there are no such requirements. The positions of different versions of same object in the logical storage space is used to indicate the relationships of the versions of same object. It avoids the complex process of establishing the links between versions. It was suggested that a versionable object (a generic object, which does not have any information about its life-span) and its versions are instances of the same class, even through their formats are different. However, versions have more attributes

than the generic object, these attributes are used to describe the features of the versions, for example 'version number', the OID of the generic object, etc. This results in different type instances (with different number of attributes) belonging to same class. To overcome this problem in TVM, the versionable classes are defined by the combination of the generic class and its version type, so all instances of versionable class have same format. Each instance has its own OID. This simplifies version management.

2.3 Temporal version and its semantics

A version is a semantically significant snapshot of a modelled object. A temporal version is the combination of version, its identifier and time information during which the version exists.

In TVM, temporal semantics [Allen 1983] are adopted; this defines the temporal primitive, temporal connectives and other related basic components.

The temporal versions for schema and the object-instance are treated in a uniform way. It is declared by a class, *temporal version,* which is used to define different types of temporal versions. The notion of temporal version is defined by a pair containing the *versioned object* and its *temporal version definition.* The versioned object can be a schema, a class, or several classes, or part of a class, i.e. a property or operation of interest.

A temporal version definition determines the type of temporal versions. It could be *historical* versions, which are used to describe an versioned object at various time; an object may have a number of *alternative* versions which are valid at same time. Each alternative version may be changed over time, therefore, the *alternative-historical* versions are also required.

2.3.1 Multiple dimensional temporal versioning

Time types of *valid time* and *transaction time* are supported in TQuel [Tansel 1993] and TSQL2 [Snodgrass 1994]. However, only one *time dimension* of each *time type* (such as valid time, transaction time) is supported. The *when* clause is the valid-time analogue of the *where* clause of SQL, and the *as-of* clause is for transaction time. For example,

retrieve (Obj.all)
where Obj.Client = "Melanie"
When Obj.**overlap** | 12PM Dec. 30, 1991|
as of present

It implies that *present* is *transaction time*, and '12PM Dec. 30, 1991' is *valid time*. To restrict support only one time attribute of each time type by the temporal database model will limit its application.

In order to describe temporal information accurately and precisely, a number of *time type*s have been introduced, apart from *valid time, transaction time.* Other type of time include, *event time* [Chakravarthy 1994], *abstract time* [Dayal 1992], *user-defined time* [Snodgrass 1994]. Each of these reflect a different angle on temporal information. A user from a different point of view could define a different time type. Moreover, there may be more than one *time attribute* of same *time type* that need to be modelled. Many business, medical and scientific applications require multiple dimensional time support - in particular, multiple valid time dimensions [Kokkotos 1995][Lü 1995].

In TVM, multiple dimensional temporal attributes are supported; these attributes could belong to different time types and for each time type arbitrary time attributes are allowed. We consider that temporal information may be viewed as existing in a multiple dimensional time space. At the conceptual level, all time dimensions are treated in the same way regardless of whether it is imposed by the computer system. It is required to indicate the time type and the name of the time attribute when it is being used. For example, there are two time attributes (TreatmentTime and ApprovalTime) in GP17-form (a Scottish dental remuneration claim form), both of which can be regarded as valid time and are often used in audit query conditions. Here, valid is used to indicate its time type, TreatmentTime and ApprovalTime indicate the names for time attributes.

SELECT *
FROM GP17-Form
WHEN valid.TreatmentTime.start **before** 01/02/95 ***and***
valid.ApprovalTime **during** [05/04/96, now]

The definition of *time type* is based on the semantics of the application (the parameterized temporal selection predicate) and the temporal domain. Therefore, time type can be defined either by the database system or the database user.

To sum up:

- There can be an arbitrary number of time dimensions.
- Time dimensions are not necessarily hard-coded into the system. New time dimensions can be declared by an application as required, and different combination of dimensions can be used for constructing version definitions.
- Giving databases multiple dimensional time support has an effect on two levels. First, it means the database system should support different *time type*s, such as valid time, transaction time, and that user may define their own time dimension by giving the temporal domain and temporal semantics. Schema and object are both versionable.

2.3.2 Alternative versioning

A database schema models a particular aspect of the world, according to the requirements of application. However, there are often several ways in which we could model an application. For example, there are may three kind of locks for same model of car at same time. The idea of alternative versions is well-know in CAD and CASE application [Sciore 1991].

Alternative versions and their histories can also be modeled by TVM If more than one version of an object (or schema) is valid at a certain time, these versions may be distinguished by version-indentifiers. For example, in order to distinguish the different kinds of locks, a version identifiers are required, these different values are called *alternative*s.

2.4 Operations

A set of operations is provided to define and manipulate versions according to the above model. In this section, we give a classification of operators. The definitions for these operations [Lü 1995] are not described in detail here.

1. Operators for schema definition and modification.
2. Operators for versionable object instance creation and modification.
3. Operators for non-versionable object instance creation, deletion and modification.
4. Operators to retrieve object instances according to query specifications.
5. Operators to retrieve schema and their object instances. The result will construct a schema-instance-hierarchy.

3. Other Considerations

3.1 Temporal-Spatial Versioning

Spatial data consist of spatial objects made up of the states (nonspatial attribute values), positions (the positions with geographic space) and shapes (the geometry features of the object; which may contains line, regions, etc.). Spatial databases facilitate the storage and processing spatial and nonspatial data. However, regardless the shapes of the objects, the problems still concern the processing of the states of objects and where the objects are in a 2 or 3 dimensional geographic space. Nevertheless, this is quite similar to temporal data processing, although the spatial semantics are different from temporal semantics. Therefore, a temporal-spatial data model can be possibly established by defining a conceptual temporal-spatial space, specifying the temporal-spatial semantics and extending the object-oriented data model to capture temporal-spatial versions.

3.2 Visualising multiple dimensional data

Visualising multiple dimensional data allows viewing of the relationships between different dimensions. Three dimensional data presented by 3-D cubes has been considered in some commercial system [Bailey 1996] [Stanley 1996]. There are two solutions to visualising data existing in more than three dimensional space.

(a) first by a set of 3-D cubes. For example, to visualise a 4-D (a,b,c,d) data model, if the relationships of dimensions a, b, c and a, b, d need to be represented, then two 3-D cubes ((a,b,c), (a,b,d)) need to be used. By using the same method, a n-dimensional cube can be represented by *n* (n-1) dimensional cubes. Therefore a *n* (n>3) dimensional cube can be represented by a set of 3-D cubes.

(b) second by a set of 2-D tables. For each 3-D cube, 3 2-D tables can be used to represent the relationships between these three dimensions. Using a set of 3-D cubes to represent 4-D cube can not show the full relationships among those four dimensions, because these 3-D cubes only show partial relationships between those four dimensions. The conclusion obtained from above is that to visualise *n* (n>3) dimensional data, 3-D cubes or 2-D tables can only show partial relationships between

the *n* dimensions. To use 3-d cubes and 2-D tables at same time provides a flexible way to show the relationships between different dimensions. A further investigation of the methods of visualising the full relationships of *n* dimensional data in two dimensional space is needed.

4 Summary and Future Work

We introduced a new conecpt - temporal versioning semantics, which provide a uniform way to understand and process the states of objects and their schema. A data model (TVM) supporting temporal versioning semantics is proposed by extending an object-oriented data model. It is able to model temporal and alternative versions, and supports multiple dimensional time.

Other issues are also under investigation in this study, they are: logical storage structures and access mechanisms for temporal data; handling schema history and the schema-instance-hierarchy; problems due to object migration and object update, etc. [Lü 1995].

Currently, we work on the construction of a query language for TVM and prototype TVM on the top of an OODBMS (Objectstore) by adding an interpretative layer between TVM and Objectstore.

References

[Agrawal 1991] Agrawal R. Buroff R. Object Versioning in Ode. In *Proceedings of the 7th IEEE Conference on Data Engineering*, 446-455, 1991

[Allen 1984] Allen, J. F. Towards a General Theory of Action and Time, Artificial Intelligence, 23, 1984

[Bailey 1996] Bailey A. Universal Data Management. In the Proceedings of 14th British National Conference on Database, Edinburgh 1996

[Barclay 1991] Barclay P. J. and Kennedy J. B Regaining the Conceptual Level in Object Oriented Data Modelling. In Aspects of Databases (Proceedings of 9th British National Conference on Databases). 1991

[Barclay 1993] Barclay P. J. and Kennedy J. B Viewing Objects. In Advances in Databases (Proceedings of British National Conference on Databases, 1993). Springer Verlag (Lecture Notes in Computer Science Series), 1993

[Bertino 1996] Bertino E., Ferrari E., and Guerrini G. A Formal Temporal Object-Oriented Data Model. In the proceedings of ETDB'96.

[Chou 1987] Chou H., Kim W. Versions and Change Notification in a Object-Oriented Database System. MCC Technical Report Number: ACA-ST-329-87, 1987

[Chakravarthy 1994] Chaakravarthy S. and Kim S. Resolution of Time Concepts in Temporal Databases. *Information Scineces*, 1994.

[Dayal 1992] Daya U and Wuu G. A Uniform Approach to Processing Temporal Queries. I the Proc. VLDB'92, 1992

[Dearle 1989] Dearle A., Connor R., Brown F. and Morrison R. Napier88 - A Database Programming Language ? In Proceedings of DBPL-2, 1989

[Kline 1993] Kline N. An Update of the Temporal Bibliography. *SIGMOD Record*, 22(4), 66-80, 1993

[Kokkotos 1995] Kokkotos S. On the Issue of Valid Time (s) in Temporal Databases, SIGMOD Record, Vol. 24, No. 3, 1995

[Lü 1995] Lü J. Temporal versioning. Working Report DCS/JL2, Napier University, Edinburgh, 1995

[Mckenzie 1991] Mckenzie L. E. Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, Vol. 23 No.4 December 1991

[Sciore 1991] Sciore E. Multidimensional Versioning for Objecy-oriented Databases. In *Proceedings of Second International Conference on Deductive and Object-Oriented.* Database, 1991

[Snodgrass 1994] R. Snodgrass A TSQL2 Tutorial. *SIGMOD RECORD*, Vol. 23, No. 1, 1994

[Soares 1995] Soares L. F. G. Nested Composite Nodes and Version Control in an Open Hypermedia System. *Information System*, Vol. 20 No. 6 pp. 501-519, 1995

[Stanley 1996] Stanley N. Microsoft Database Technologies - An Inside View. In the Proceedings of 14th Brithsh National Conference on Database, Edinburgh 1996

[Tansel 1993] Tansel A. Z. Temporal Databases, Benjiaming/Cummings, 1993