Temporal Versions Model

Mirella M. Moro, Nina Edelweiss, Clesio S. dos Santos
Universidade Federal do Rio Grande do Sul, Instituto de Informática
Caixa Postal 15064, CEP 91501-970, Porto Alegre, RS, Brazil
{ mirella, nina, clesio } @inf.ufrgs.br

Abstract

This work presents an alternative for the union of temporal data and a version model. The result, the Temporal Versions Model, is able to store object versions and, for each version, the history of its dynamic properties values. TVM is ideal for modeling time-evolving systems that need to manage design alternatives as versions. One of the main features of our model is the possibility of having two different time orders, branched time for the object and linear time for each version. The model supports integration with existing databases, by allowing normal classes among the temporal versioned classes. Finally, an approach to its implementation on top of a commercial database within an integrated environment is presented.

Keywords: Temporal database, Versions model, time-evolving systems specification

1 Introduction

In design applications, different alternatives or versions of a project are kept in the database. Historically, first researches about versions are related to the areas of CAD (Computer Aided Design), CASE (Computer Aided Software Engineering), and SCM (Software Configuration Management) [5, 9, 20, 28]. Nowadays, version concepts are spread over other domains, such as concurrent projects, schema evolution, XML, and document versions [1, 4, 21].

A version describes an object in a period of time or from a certain point of view. Although some design alternatives are stored as versions, not all the history of data chances is recorded. Important modifications may have occurred whose related data are lost. The full history is only accessible if a temporal database is used.

A temporal data model specifies both static and time-varying aspects of the application. By definition, a temporal database must follow the principle that all data excluded by the user is kept in order to preserve the complete data history. Many temporal models have been researched and specified in the past two decades, most of them just extending conventional data models with temporal features [6, 7, 11, 24, 25, 26, 27].

Handling versions (joining, separating, discarding and justifying wrong possibilities) tends to decrease costs and improve quality of final products. In this context, presenting versions with their respective temporal aspects has the advantage of getting design information related to specific periods. The objective of this work is to bring together both versioning and temporal features. The database stores the versions of an object and, for each version, the history of data changes made in the values of its dynamic attributes and associations. This model is called Temporal Versions Model (TVM).

TVM differs from other models because it presents two time orders (branched time in objects and linear time in versions) and allows the user to define which properties will have their history stored. This second feature is important for limiting the data space, allowing a better performance. Furthermore, TVM has two important aspects: the user can specify the system considering design alternatives as well as data history evolution; and he can merge TVM classes with existing specifications, as the Model does not require that all classes are temporal versioned.

The use of a powerful temporal data model does not require a specific database management system (DBMS) that supports it to implement an application. A tendency is to implement the specific model on top of a conventional database through mapping of temporal information to explicit attributes. Some experiences with mappings have been done and showed their viability [11, 26, 27].

Thus, as an additional objective, an entire environment for supporting TVM is designed [14]. Specifically, the mapping from the TVM base hierarchy to an object-relational database and its implementation on top of a commercial database are detailed. The mapping is divided in two parts: the representation of the model's hierarchy, and the mapping of the application classes [15]. Generic algorithms for mapping all operations from the TVM base hierarchy to the underlying DBMS are detailed in reference [14]. Due to space limitations, this work presents a summary of this mapping process and some interfaces for temporal versioned class definitions, which create tables in a conventional database according to the TVM specification.

This paper is structured as follows. Section 2 briefly introduces some concepts of versions and time. Section 3 defines the Temporal Versions Model. An environment for TVM on top of a commercial database is presented in section 4. Section 5 illustrates a system specification using TVM. Section 6 cites some related works. Finally, section 7 summarizes the main ideas of the paper, discusses some future topics and shows other research issues that are based on the work presented here.

2 Basic Concepts

This section briefly introduces the concepts of versioning and time. The concept of *version* enables the user to keep different design alternatives. Versions can be defined as distinct snapshots of an object under development, in different states, which share some identifiable common characteristics. Considering an object-oriented database, there are two types of versions, according to the level in which versioning is applied: *class versioning* (evolution of classes) and *instance versioning* (modifications of the properties inside the instances). Only the second type of versioning is considered in TVM.

Instance versioning implies that different versions of the same object differ only by the value of some of their properties. However, only versioning is not enough to store all history changes in instances with versions. Only those values that are identified explicitly by the user as a new instance version are kept. The addition of the temporal dimension enables the storage of the whole evolution of attribute and association values of the instances. Hence, the proposed model (TVM) is based on concepts of instance versioning and time, allowing the storage of object versions and, for each version, its lifetime and the history of all the changes made on its elements.

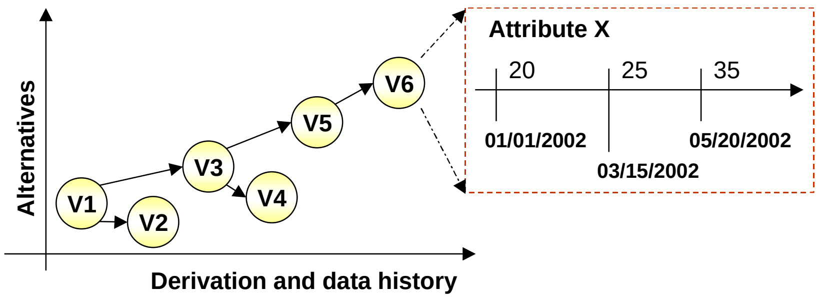


Figure 1. Alternatives X Derivation, and data history

Figure 1 presents the evolution of one generic versioned object, and how the derivation of the alternatives (*V1*, *V2*, *V3*, *V4*, *V5*, and *V6*) are kept. Child versions represent design derivations from a parent version, and sibling versions represent design alternatives. As this can overload the storage capacity, the user can choose the elements for which he wants the whole history stored by defining them as *temporal*. Furthermore, new versions are defined explicitly by the user, which means that the system keeps changes inside the version history and creates a new version only if the user requests it.

Figure 1 also shows the time-evolving behavior of a temporal attribute (X). Its value is 25 from 01/01/2002 on. Then, it changes to 25 on 03/15/2002, and finally, to 35 on 05/20/2002. All these values are stored in the attribute history inside one object or version. In this figure, the attribute history is inside version V6. The other versions have the same attribute X, but with other values. So, the same attribute may have different values for the same versioned object, representing the branched time order.

As data evolves, values may be associated with different timestamps. *Valid intervals* store the history of an evolving reality, while *transaction intervals* store the sequence of temporal classes states in the database. In bitemporal databases, all temporal values are associated with both intervals.

3 Temporal Versions Model

In this section, TVM characteristics are presented separated in temporal features and versioning features. A summary of the TVM class definition language is also shown.

3.1 Temporal Features

In TVM, time is associated to objects, versions, attributes and associations, allowing a better and more flexible modeling of reality. The user can define attributes and associations as *static* (when they do not have the values variation stored) or *temporal* (all the values changes are stored creating their histories). A class may present attributes and associations of both types.

A temporal versioned object has a timeline for each of its versions. As more than one version of the object may be available at the same time, two different time orders are supported by TVM: (i) branched time for an object, due to its different versions timelines, and (ii) linear time within each version. Time varies in a discrete way, and temporality is represented in the Model through temporal intervals with bitemporal timestamps. Temporal attributes and associations are associated with the temporal labels, for valid and transaction intervals.

3.1.1 Logical and physical removal operation

A removal operation can be of two forms, *logical* or *physical*. When a version is logically removed, it is moved to the deactivated status and has its lifetime finished. If there are temporal attributes or associations, their final valid times receive the object or version final lifetime at the moment of the logical removal operation.

The physical removal, also known as *vacuuming*, may be exceptionally executed, and aims to remove from the database all information that is not relevant to the application anymore. It is performed when necessary to reduce the database size, which may increase very much when all temporal data is kept [6].

Support for transaction time brings with it the potential for accessing any past database state, without having to execute recovery processes. Physical deletion limits this potential, and while this may be desirable, care should be taken to avoid an adverse impact on the utility of the data kept in the database [12][1]. TVM does not define the rules for this type of removal operation, assuming that all removal operations are logical.

3.2 Versioning Features

Versions of a real world entity must be kept together. Grouping versions of the same object constitutes a *versioned object*. Class instances are one of the following: a versioned object, a version, or an object without versions, as illustrated in Figure 2. Versions in versioned object are related through a derivation relationship, defining a *directed acyclic graph*. Each versioned object has a *current version*, which is used whenever operations are applied to a versioned object

[1] The impact of allowing physical deletion in the database and the query language are clearly described by Jensen in [12].

without specifying one of its versions. The current version is defined by the system as the most recently created. The user may also set a different version to be the current one.

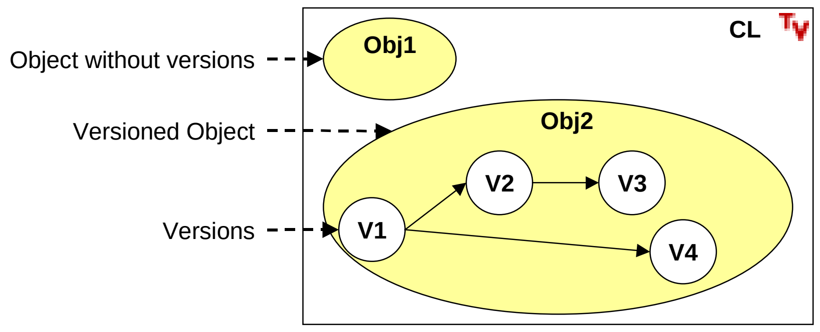


Figure 2. TVM instances: an object without versions, a versioned object, and four versions

3.2.1 *Classes Hierarchy*

Figure 3 illustrates the TVM base class hierarchy. Basically, TVM allows two types of classes: (i) standard classes, for which neither versions nor temporal features are available (defined as *Object* subclasses), and (ii) temporal versioned classes, supporting both features (defined as *TemporalVersion* subclasses). The class *TemporalObject* has methods to manage temporal labels. The class *TemporalVersion* has the version attributes *configuration* and *status* (that informs if the version belongs to a configuration and its status, respectively) and the navigation attributes *ascendant*, *descendant*, *predecessor*, and *successor*. Finally, the class *VersionedObjectControl* has attributes to report the current version, the number of configurations and versions, the first and last versions, the next version number, and whether the user has specified a current version or not. All classes have methods to get and update the attributes and methods with specific functions detailed in reference [14].

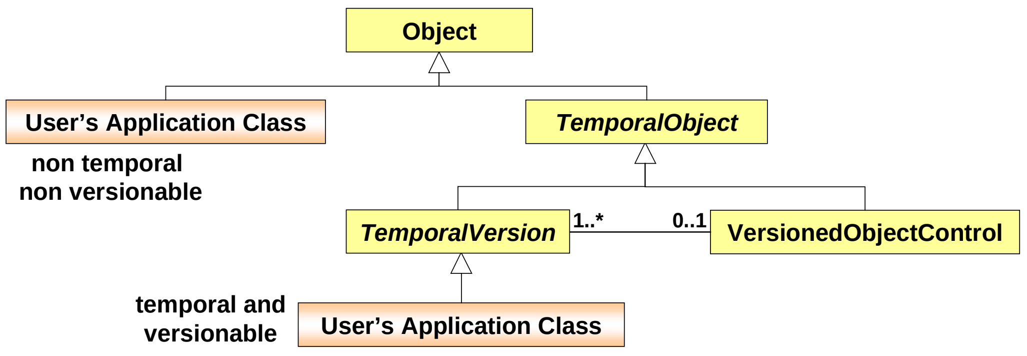


Figure 3. TVM base class hierarchy

3.2.2 *Version States*

A version changes its status during its lifetime (Figure 4). A version is created in the state *working*. In this state the version is essentially temporary and can be derived, modified, queried, and removed. When a version is derived from another one, its predecessor is automatically promoted to *stable*, avoiding modifications on versions that are important from the historical point of view. In this state the version cannot be modified, but can be derived, promoted to *consolidated*, queried, and removed. In the c*onsolidated* state the version can be queried and derived, but cannot be modified nor removed. TVM assumes only logical removals, when the version is moved to the *deactivated* status and has its lifetime finished. In this state, versions can only be queried, or restored to the original status. A summary of operations in each state is presented in Table 1.

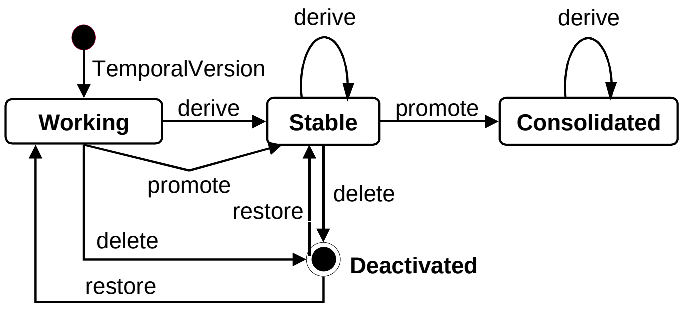


Figure 4. Status and events that cause transitions

Status / Operations	working	stable	consolidated	deactivated
Derive	**Y**	**Y**	**Y**	-
Promote	**Y**	**Y**	-	-
Modify	**Y**	**N**	**N**	**N**
Exclude	**Y**	**Y ***	**N**	-
Query	**Y**	**Y**	**Y**	**Y**

Legend: **Y** *enable* **N** *unable* **-** *no defined*
***** *only if the version has no successor*

Table 1. Status and allowed operations

3.3 Class Definition Language

A class definition language for TVM is completely defined in [14], with a simplified syntax presented in Figure 5. The clause *HasVersions* specifies temporal versioned classes. The clause *temporal* indicates if the property will have its evolution stored. A class may be temporal or common aggregation of other classes. The specification of the inverse association indicates if the relationship has two modes of navigation.

```
Class className [ HasVersions ] [ inherit className ]
  [[ temporal ] aggregate_of [n] className ( by value | by reference )
  {, [ temporal ] aggregate_of [n] className ( by value | by reference )} ]
  ( [ Properties:
      [ temporal ] attributeName : attributeDomain ;  { [ temporal ] attributeName : attributeDomain ; } ]
    [ Relationships:
      [  temporal ] relationshipName (1:1 | 1:n | n:1 | n:n) [ inverse inverseRelationshipName ] relatedClassName ;
       {[ temporal ] relationshipName (1:1 | 1:n | n:1 | n:n) [ inverse inverseRelationshipName ] relatedClassName; } ]
      [ Operations: { operationDefinitions } ] ) ;
```

Figure 5. Simplified definition language syntax

TVM has many other features, such as [14]:

- Temporal integrity rules related to the object lifetimes and management of temporal labels.
- Temporal types hierarchy.
- Restoring of deleted objects.
- Extension inheritance, in which versions are enabled in different hierarchy levels.
- Configuration, which assures only one version (or object) in each inheritance level per entity.

4 TVM on top of a Commercial DBMS

The approach of implementing TVM on top of a conventional DBMS is adopted in this work. Existing commercial DBMSs can be used as long as a proper mapping from the temporal data model to the data model underlying the adopted DBMS is provided. The database DB2 was chosen due to its support of temporal data types similar as the standard SQL-92 [24].

This section presents the mapping that is divided in two parts: the representation of the model's hierarchy, and the mapping of the application classes. Finally, some interfaces of Temporal Versions Model Environment are illustrated.

4.1 The Model Hierarchy Mapping

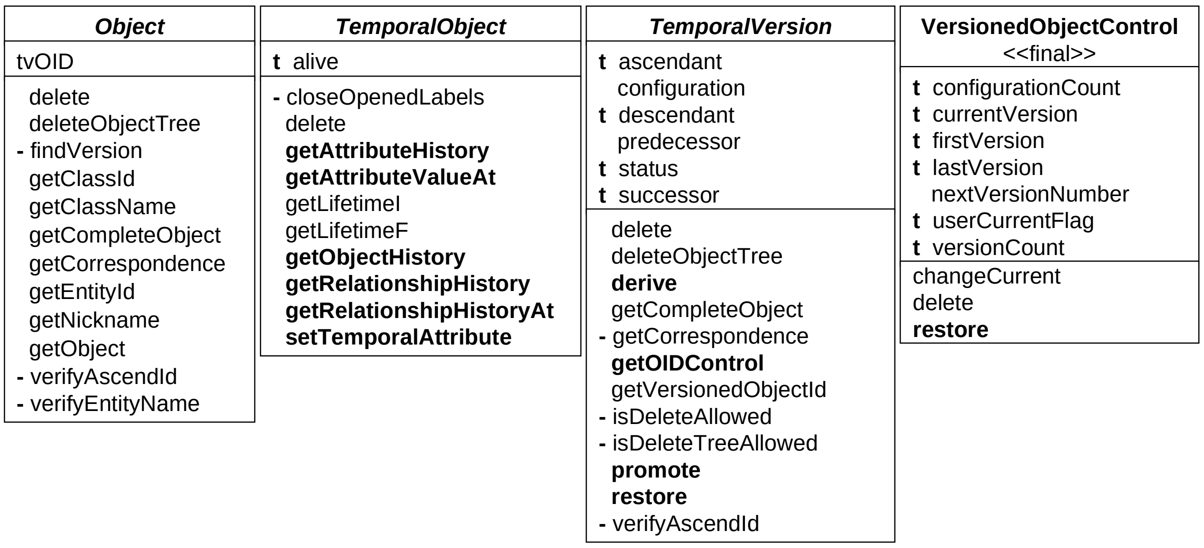


Figure 6. The classes of the hierarchy with their attributes and their most important methods

The class hierarchy is mapped to an equivalent type hierarchy of DB2, in which some methods can be executed through SQL commands and queries. Figure 6 illustrates the attributes and the most important methods defined for the model hierarchy. Basically, only the methods related to time and version features need to be translated.

The methods mapping is completely presented in reference [14] in form of generic algorithms, which combine value definitions, variables, procedure calls, conditions, loops, SQL queries, and possible errors. Additionally, questions about access rights to the *VersionedObjectControl* instances must be specified by the database administrator. In summary:

- Class *Object* is mapped to a non instantiable structured type.
- Class *TemporalObject* is mapped to a non instantiable structured type with the attribute *alive*. This class presents a method *delete* that is mapped to a stored procedure that receives the OID and executes the logical exclusion by changing the status to *deactivated*, registering the attribute *alive* value as *false*, associated to the present instant.
- Class *TemporalVersion* is mapped to a non instantiable structured type with its attributes. It has the methods for handling version status.
- Class *VersionedObjectControl* is mapped to a structured type with its attributes and its methods that manage the versioned objects.

4.2 The Application Classes Mapping

Because DB2 does not implement collections, this part of the mapping requires all classes in the first normal form. For each standard class, the following objects must be created in the database: (1) a structured type with the same class structure, to which is added the attribute *ascendant* in case of the class is a subclass; and (2) a table to store the class instances, called *main table*. If this last table stores the instances of a subclass, two triggers shall be associated to the *insert* operation. The first one verifies if the ascendant value corresponds to a valid OID, and the second one forbids the user to inform values to the inherited attributes.

For each temporal and versionable class, the same structured type and table mentioned above are created, adding the following:

- A table for each temporal attribute and association to store its history, called *auxiliary table*.
- A trigger associated to the temporal attribute and association updates, which stores the old values in its respective auxiliary table. For each new value inserted in the auxiliary table, (1) if the initial valid time (of the new value) is not informed by the user, as usual, it assumes the present moment; and (2) the final valid time (of the old value) receives the initial valid time (of the new value) minus one instant.
- A trigger associated to the operation *update*, which allows the user to modify only versions in the status *working*.
- A trigger associated to the operation *update* of the *VersionedObjectControl* attributes, which do not allow the user to modify the attributes manually.

4.3 TVM Environment

The TVM Environment has a set of interfaces for modeling classes, handling and querying instances. It also hides from the user the implementation and mapping details necessary for TVM to work on top of the database.

Specifically, the user specifies the classes in a definition tree through the interface for TVM Class Definition. The user can also map the specification to the TVM Definition Language and to SQL, in order to store the system in the underlying database. Table 2 presents the meaning for the icons on the specification tree, and Figure 7 presents this interface and part of a simple example.

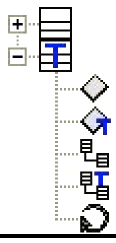
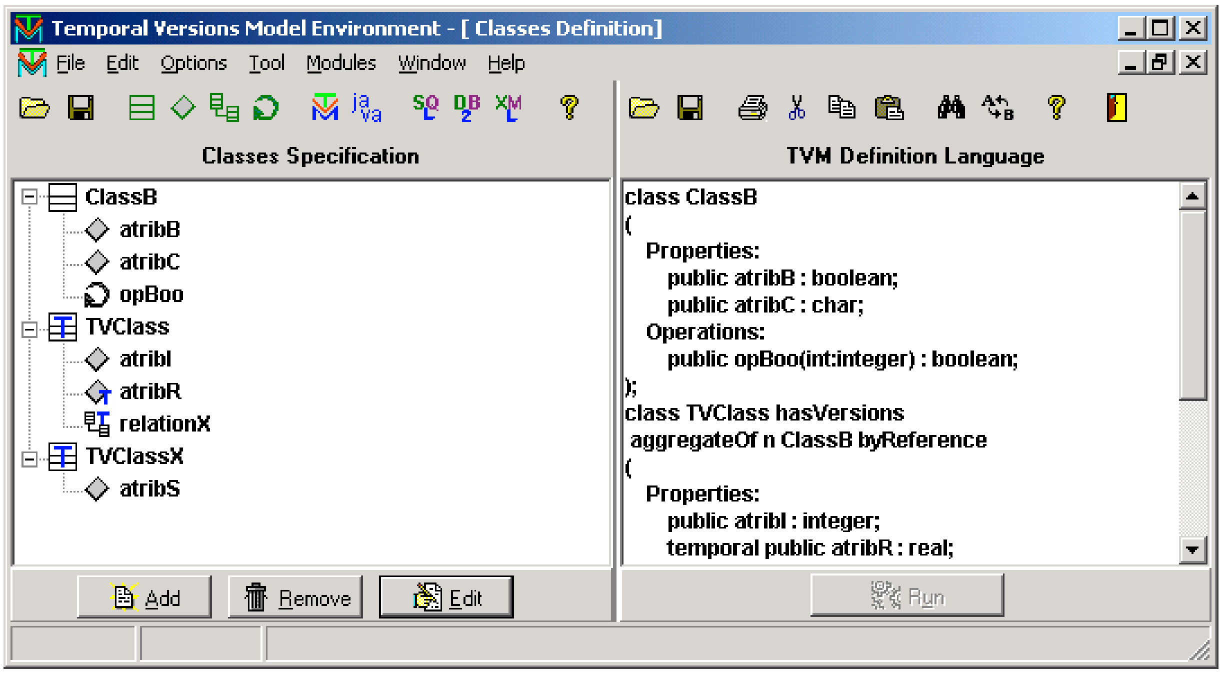
Icon	Meaning
	Normal class
	Temporal class
	Attribute
	Temporal attribute
	Association
	Temporal association
	Operation

Table 2. Icons on the specification Tree


Temporal Versions Model Environment - [Classes Definition]
File Edit Options Tool Modules Window Help
Classes Specification
TVM Definition Language
Add
Remove
Edit
Run

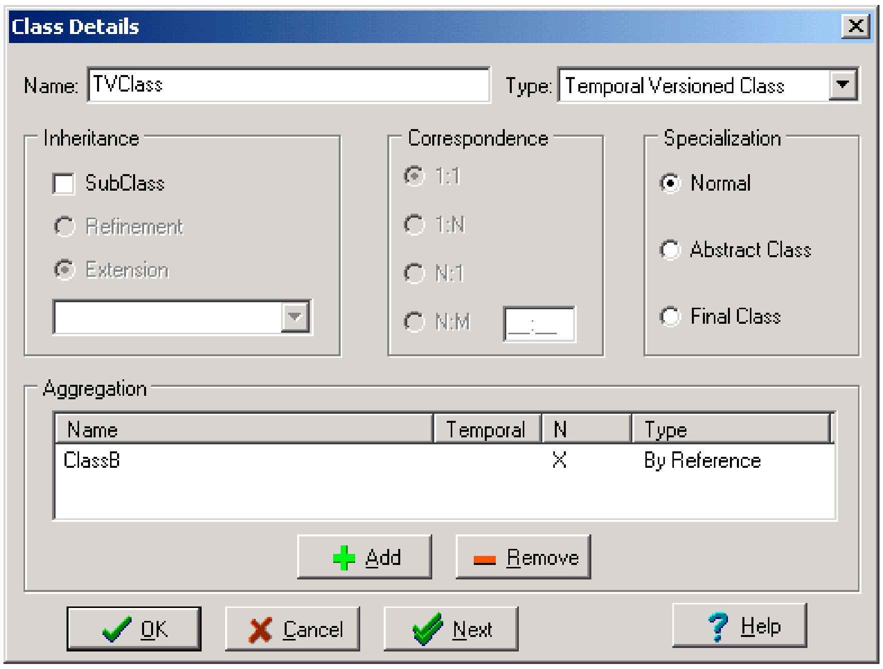
ClassB
- atribB
- atribC
- opBoo

TVClass
- atribI
- atribR
- relationX

TVClassX
- atribS

```
class ClassB
(
  Properties:
    public atribB : boolean;
    public atribC : char;
  Operations:
    public opBoo(int:integer) : boolean;
);
class TVClass hasVersions
 aggregateOf n ClassB byReference
(
  Properties:
    public atribI : integer;
    temporal public atribR : real;
```

Figure 7. TVM Environment: Classes Specification and TVM Definition Language



Name: TVClass
Type: Temporal Versioned Class

Inheritance: [ ] SubClass; Refinement [✓ Extension]

Correspondence: [✓ 1:1] 1:N N:1 N:M

Specialization: [✓ Normal] Abstract Class Final Class

Name	Temporal	N	Type
ClassB		X	By Reference

Figure 8. TVM Environment: screen for Class Details specification

Figure 8 illustrates the screen for Class Details specification. It requires the class name; the type (whether temporal versioned or not); the inheritance (by extension between temporal versioned class, and by refinement between standard classes); the correspondence that is enabled only if there is an inheritance by extension; whether the class is abstract, final or none; and if the class is an aggregation (by value or by reference) of other classes. The environment has screens for specifying attributes, relationships and operations, which follow this same pattern.

5 Illustrative Example

To illustrate the presented concepts, TVM is used to model a simple application, a website design company. Beyond its clients' sites, the company keeps the professional pages of its employees. Only the nucleus of the model is presented in Figure 9. The specification using the TVM class definition language is presented in Figure 10.

Each Website is composed by one or more pages, one of them is the initial or main page. Each site is associated to a page pattern, which defines the background color and image, a banner, and the default font specification. The pattern is used as a standard for the company's page layout. Following to the company instruction, this pattern varies according to the seasons of the year and commemorative dates. For instance, during the company's anniversary month, the background image presents a cake with lighted candles, and the banner offers special discounts.

The features of version and time are used in the following:

- The *WebSite*, *WebPage*, and *PagePattern* classes - all the alternatives of the site, its pages and pattern, as well as the temporal attributes and associations values are stored.
- The relationship *associatedWith* - the main relationship of the model through which the company changes its website pattern.
- The attribute *banner* - each pattern can be associated with the website for a long time, hence several banners can be used during the same period.
- Relationship *initialPage* - the evolution of the initial or main page is also stored.

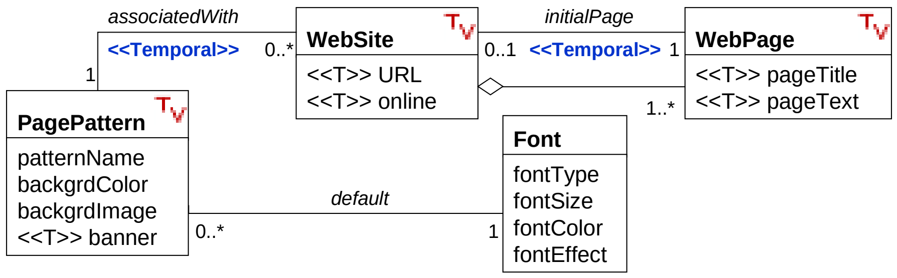


Figure 9. Class Diagram of the example

```
Class WebSite HasVersions
 aggregate_of n WebPage by reference
  ( Properties:
        temporal URL : string ;
        temporal online : boolean;
   Relationship:
        temporal initialPage 1:1 WebPage;
        temporal associatedWith 1:1 PagePattern; ) ;
Class WebPage HasVersions
 ( Properties:
        temporal pageTitle : string;
        temporal pageText : text; ) ;

Class PagePattern HasVersions
   ( Properties:
          patternName : string;
          backgrdColor : string;
          backgrdImage : string;
          temporal banner : string;
   Relationship:
          default 1:1 Font;  ) ;
Class Font
  ( Properties:
          fontType : string;
          fontSize : integer;
          fontColor : string;
          fontEffect : string;  ) ;
```

Figure 10. Class definition language

Among the advantages of using TVM to model this application, the designer simplifies the tasks of changing the pattern associated with all pages (which can be done by adding a new value on the relationship *associatedWith*) and keeps the period of each alternative stored.

Additionally, he assures that the evolution of the clients' website is stored. If a client wants an old page to be restored, the user has only to modify the valid time of the page.

The company has many uses for the information stored in the database built from this modeling. One of the most important is the chance to discover patterns and clients' profiles by using data mining techniques, for instance.

This example uses versioning to illustrate the different versions according to the time in which they are valid. The same model could be used to specify other aspects. For instance, the designer could create different versions according to the technology present inside the pages. Hence, the pages could have versions with HTML, java script, asp, flash, and so on.

Figure 11 illustrates the *PagePattern* class with its versioned objects *Autumn*, *Winter*, *Spring*, and *Summer*. The versioned object *Winter* is detailed with its five versions. The first one (9,15,1) is the default page for the Winter. The second (9,15,2) and the third (9,15,3) ones were derived as alternatives for the Christmas. The fourth (9,15,4) one is the version for the new year, and the last one (9,15,5) was derived from it as the new millennium version.

Figure 12 presents the evolution of the *associatedWith* relationship for one employee's website graphically. The *WebSite* version is always the same one (9,8,1), while the association with the pattern changes according to the commemorative date inside the Winter period.

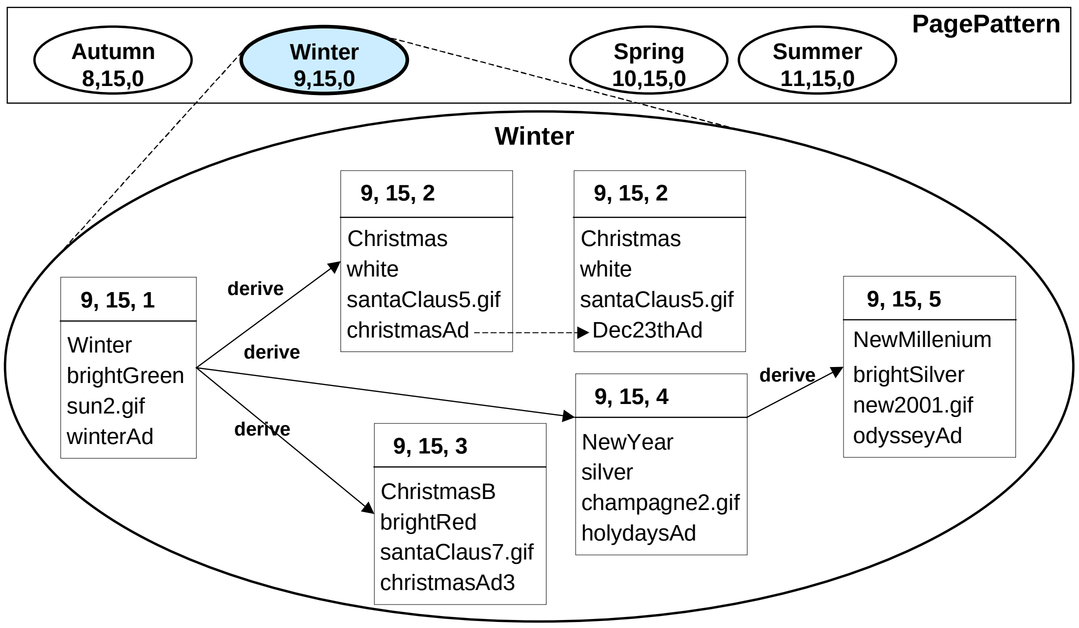


Figure 11. Graphical representation of one versioned object. The change in the attribute value of *banner* (*christmasAd* to *Dec23thAd*) does not create a new version, only gives birth to one more value in its history

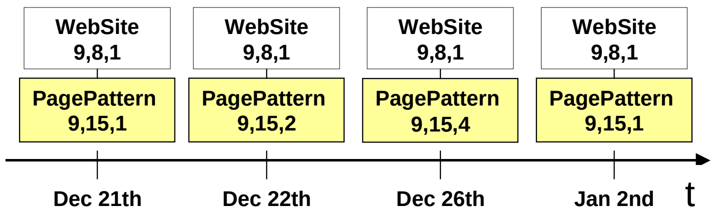


Figure 12. Temporal relationship associatedWith timeline for one website in the Winter period

6 Related Work

Different application areas that need support on evolutionary development, such as CAD and CASE tools, motivated versions-related research. More recently, the concept of version has been spread to other areas, such as concurrent projects, schema evolution, XML, document versions, and hypermedia systems [2, 4, 8, 21]. A good survey on versions model for software configuration management can be found in reference [5]. In addition, reference [10] presents a classified list of different kinds of versioning applications, and specific summaries with versioning in hypermedia area.

The concept of time may be present not only in temporal database modeling but also in queries, constraints, real-time applications, multimedia temporal synchronization, active databases, deductive databases, and so on. About temporal databases modeling and implementation, a good set of works is grouped by Tansel (et al) in reference [25]. Jensen also grouped his vast set of published and non-published papers about temporal data, query, implementation and design in reference [12]. Implementation issues are analyzed by Snodgrass in reference [24].

These two concepts, version and time, are mostly treated individually in the literature, which abounds with models for versioned information [2, 3, 5, 9, 13, 20, 28] and temporal data [6, 12, 22, 23, 24, 25]. The novelty of this paper is to put them together, with equal and simultaneous treatment.

This idea appears previously in an extension proposed by Wuu and Dayal for the OODAPLEX model, as a uniform model for temporal and versioned object-oriented databases [28]. They rely on the inherently rich type system of OODAPLEX in order to model temporal information. Time points are treated as abstract objects, and a type hierarchy of time types is defined to support various notions of time (including versions). Several temporal functions and constraints are defined to introduce additional time-related semantics to the system. The retrieval and manipulation of temporal and non-temporal information is uniformly expressed. However, the concept of version is not defined explicitly.

Rodrigues, Ogata and Yano propose a formal definition of a temporal versioned Object-Oriented data model, called TVOO [20]. They focused on the construction of a formal framework that can be tailored to the needs of a given time varying system. All objects and relationships between them are time dependent, and the history of the changes is kept in versions hierarchies. Every object update results in a new version of that object. An evolution history is kept for each class as a set of versions, where each object instance has its corresponding version hierarchy. Versions behave independently of each other, have a lifespan, and are typed as a partially ordered tree.

The main differences and similarities among TVM and these models are the following:

- Basic structure – TVM proposes a hierarchy as the basis for the user's class specifications, while TVOO uses a formal framework, and the OODAPLEX extension works with a hierarchy of abstract time types.
- Objects – TVM and the OODAPLEX extension allow temporal versioned objects among normal objects, while TVOO specifies that all objects are temporal.
- Versioning – each change generates a new version in TVOO, while in TVM the user must create explicitly new versions.

- Time dimensions – TVM and the OODAPLEX extension have both transaction and valid time, while TVOO has only transaction time.
- Derivation hierarchy – all models have concepts to manage the derivation order, but while this hierarchy is represented by a tree in TVOO, it is represented by a directed acyclic graph in TVM and in the OODAPLEX extension.
- Exclusion – the three models have logical exclusion, and excluded objects cannot be recreated in TVOO.

7 Summary and Concluding Remarks

The Temporal Versions Model is the union of a versions model with temporal information. TVM is ideal for modeling time-evolving systems that need to manage design alternatives as versions. It offers three main advantages: design alternatives storage and management; history of evolving data management; rebuilding of a database state in any past date, without using complex operations as backup and recovery. Besides, the Model differs from others due to extension inheritance, branched time order (not usual in the models found on the literature in which the time order is almost always linear), integration with existing specifications (it allows classes without time and versions among temporal versioned classes), and homogeneous treatment for valid and transaction timestamps.

The Model is presented with rich details in [14], which include: temporal integrity rules; logical exclusion; temporal types hierarchy; state diagram for versions; base class hierarchy with attributes, associations, operations and working process; extension inheritance; configuration; possible associations between standard and temporal versioned classes; class definition language. A study comparing TVM with other models is also presented.

This work summarizes the mapping from the TVM base hierarchy and application classes to an object-relational database and details the tool for support user's classes specification. The user can specify classes, attributes, associations and operations in this interface without previous knowledge of the TVM language. The tool maps the specification to TVM definition language and generates the script for creating the respective tables on the database. A complete architecture for keeping hidden the implementation details from the user is proposed in [14]

Some questions and improvement suggestions are still open, such as: language support for new literal and structured type specifications; data manipulation language; rules for ensuring the ACID properties (atomicity, consistency, isolation, and durability); operations for generating configurations, and creating an entity based on an existing entity.

This work is part of a project that aims to implement an integrated software environment for class specification, object versioning, versions management, query, and visualization. The phase of TVM specification and implementation is almost complete. Publications about this work are presented in references [15, 16, 17, 18]. This work finishes the project's first phase. Our research group keeps working with time and versions, and some results are already published.

Based on the TVM definition, a query language is defined in reference [19]. This language is being mapped to SQL in order to run on top of mapped tables. A complete data manipulation language (with create, insert, update and delete statements) is also under development. Furthermore, the same project is extending TVM to support schema versioning and evolution, with a plan already presented in [8, 21].

Other future research topics are: comparative analysis between a standard database and another with the TVM features; query optimization and indexing for a better performance; test of tools for object oriented modeling joining with TVM features; and mapping from TVM to a commercial object oriented DBMS.

References

[1] Al-Khudair, A. et al., “Object-Oriented Versioning in a Concurrent Engineering Design Environment”, *Proc.18th British Nat’l Conf. Databases* (BNCOD), England, 2001, pp.105-125.

[2] Agrawal, R. et al., “Object versioning in Ode”, *Proc. 7th Int’l Conf. Data Engineering* (ICDE), IEEE Computer Society, Tokio, 1991, pp. 446-455.

[3] Beech, D. and Mahbod, B., “Generalized Version Control in an Object-Oriented Database”, *Proc. IEEE 4th Int’l Conf. Data Engineering*, USA, 1988, pp.14-22.

[4] Chien, S-Y., Tsotras, V.J. and Zaniolo C., “Efficient Management of Multiversion Documents by Object Referencing”, *Proc. 27th Int’l Conf. of Very Large Data Bases* (VLDB), Italy, 2001, pp. 291-300.

[5] Conradi, R. and Westfechtel, B., “Version Models for Software Configuration Management”, *ACM Computing Surveys*, vol. 30, no. 2, June 1998, pp. 232-282.

[6] Edelweiss, N., Hübler, P., Moro, M.M. and Demartini, G., “A Temporal Database Management System Implemented on Top of a Conventional Database”, *Proc. 20th Int’l Conf. the Chilean Computer Science Society* (SCCC), Chile, 2000, pp. 58-67.

[7] Etzion, O., Jajodia, S. and Sripada, E., *Temporal Databases: Research and Practice*. Springer-Verlag, LNCS 1300, 1998.

[8] Galante, R. M. et al., “Dynamic Schema Evolution Management using Version in Temporal Object-Oriented Databases”, *13th Int’l Conf. Database and Expert Systems Applications* (DEXA), France, 2002, to be published.

[9] Golendziner, L.G. and Santos, C.S. dos., “Versions and configurations in object-oriented database systems: a uniform treatment”, *Proc. 7th Int’l Conf. Management of Data* (COMAD), India, 1995, pp. 18-37.

[10] Hicks, D.L. et al., “A Hypermedia Version Control Framework”, *ACM Transactions on Information Systems*, vol.16, no.2, Apr. 1998, pp. 127-160.

[11] Hübler, P. N. and Edelweiss, N., “Implementing a Temporal Database on Top of a Conventional Database: Mapping of the Data Model and Data Definition Management”, *Proc. 15th Brazilian Symposium on Databases* (SBBD), Brazil, 2000, pp. 259-272.

[12] Jensen, C.S., *Temporal Database Management*, doctoral dissertation, Dept. of Computer Science, Aalborg University, Denmark, 2000, http://www.cs.auc.dk/~csj/Thesis/.

[13] Kim, W., Bertino, E. and Garza, J.F., “Composite objects revisted”, *Proc. ACM SIGMOD Int’l Conf. Management of Data*, ACM Press, Oregon, 1989, pp.337-347. SIGMOD Record vol. 18, no. 2, June 1989.

[14] Moro, M.M., *Modelo Temporal de Versões [Temporal Versions Model]*, master’s thesis, Inst. de Informática, UFRGS, Brazil, 2001, http://nina.inf.ufrgs.br/Dissertacao/MirellaMoro.pdf (in Portuguese).

[15] Moro, M.M. et al., “Adding Time to an Object-Oriented Versions Model”, *Proc. 12th Int’l Conf. Database and Expert Systems Applications* (DEXA), Springer-Verlag, Germany, LNCS 2113, 2001, pp. 805-814.

[16] Moro, M.M. et al., “Dynamic Systems Specification using Versions and Time”, *Proc. 5th Int’l Database Engineering and Applications Symposium* (IDEAS), IEEE Computer Society, France, 2001, pp. 99-107.

[17] Moro, M.M. et al., “A Temporal Versions Model for Time-Evolving Systems Specification”, *Proc. 13th Int’l Conf. Software Engineering & Knowledge Engineering* (SEKE), Argentina, 2001, pp.252-259.

[18] Moro, M.M. et al., “Um Modelo Temporal de Versões para Especificação de Aplicações” [A Temporal Versions Model for Specifying Applications], *Proc. 4th Iberoamerican Work. Requirements Eng. and Software Environments*,Costa Rica,2001,pp.336-347 (in Portuguese).

[19] Moro, M.M. et al., “TVQL - Temporal Versioned Query Language”, *13th Int’l Conf. Database and Expert Systems Applications* (DEXA), France, 2002, to be published.

[20] Rodríguez, L., Ogata, H. and Yano, Y., “TVOO: A Temporal Versioned Object-Oriented data model”. *Information Sciences*, Elsevier, vol. 114, 1999, pp. 281-300.

[21] Roma, A.B.S. et al., “Gerenciamento Temporal de Versões para Evolução de Esquemas em BDOO” [Temporal Versions Management for Schema Evolution on OODB], *Proc. 16th Brazilian Symposium on Databases* (SBBD), Brazil, 2001, pp.110-124 (in Portuguese).

[22] Sarda, N. L., “Extensions to SQL for historical databases”, *IEEE Transaction on Knowledge and Data Engineering*, vol.2, no.2, June 1990, pp. 220-230.

[23] Snodgrass, R.T., “Temporal Object-Oriented Databases: A Critical Comparison”, *Modern Database Systems - The Objetct Model, Interoperability, and Beyond*, ACM Press, New York, 1995, pp.386-408.

[24] Snodgrass, R.T., *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufmann Publishers, 2000.

[25] Tansel, C.G. et al., *Temporal Databases - Theory, Design and Implementation*. Benjamin/Cummings.

[26] Theodoulidis, B. et al., “The ORES Temporal Database Management System”, *Proc. ACM SIGMOD Int’l Conf. Management of Data*, ACM Press, USA, 1994, pp. 511. SIGMOD Record, vol. 23, no. 2, June 1994.

[27] TIME DB, *A Temporal Relational DBMS*. TimeConsult. http://www.timeconsult.com/ Software/Software.html.

[28] Wuu, G.T.J and Dayal, U., “A Uniform Model for Temporal and Versioned Object-Oriented Databases”, *Temporal Databases: Theory, Design, and Implementation*, A.Tansel et al. (ed) Benjamin/Cummings, 1993, pp. 230-247.