Temporal Versions Model

Mirella Moura Moro, Nina Edelweiss, Clesio Saraiva dos Santos

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – 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 and relationships values. TVM is ideal for modeling time-evolving systems that need to manage design alternatives as versions. An interface for modeling TVM classes is also presented.*

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*) [Golendziner and Santos 1995], [Conradi and Westfechtel 1998]. Nowadays, version concepts are spread over other domains, such as concurrent projects, schema evolution, XML, and document versions [Chien et al 2001].

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 [Tansel et al 1993], [Etzion et al 1998], [Edelweiss et al 2000].

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 paper 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 relationships. This model is called *Temporal Versions Model* (TVM). This work also presents the interface for temporal versioned class definition for creating tables in a conventional database according to the TVM specification.

This paper is structured as follows. Section 2 presents the Temporal Versions Model. An environment for TVM is illustrated in section 3. Finally, section 4 summarizes the main ideas of the paper and presents future works.

2. Temporal Versions Model - TVM

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. There are two types of version, 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 this paper.

Instance versioning implies that different versions of the same object differ only by values 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 to store the whole evolution of attribute and relationship values of the instances. Hence, the proposed model (TVM) is based on the 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 presents 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 [Moro et al 2001a].

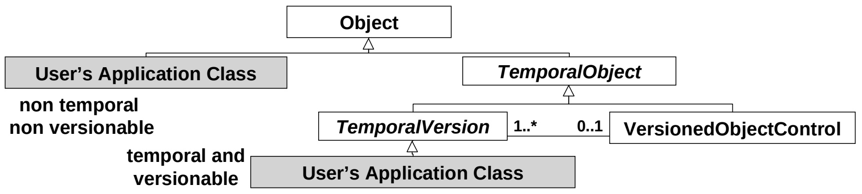


Figure 1. TVM base class hierarchy

In TVM, time is associated to objects, versions, attributes and relationships, allowing a better and more flexible modeling of reality. The user can define attributes and relationships 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 relationships 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 (*valid intervals* store the history of an evolving reality, while *transaction* *intervals* store the sequence of temporal classes states in the database). Temporal attributes and relationships are associated with the temporal labels, for valid and transaction intervals.

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. Versions in versioned object are related through a derivation relationship, defining a *directed acyclic graph*. Each versioned object has a *current version*, used whenever operations are applied to a versioned object 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.

A class definition language for TVM is completely defined, with a simplified syntax presented in Figure 2. 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 relationship 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 ; } ]
    [ Relationship:
      [  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 2. Simplified definition language syntax

TVM has many other features, such as: temporal integrity rules, graphical representation, versions states (*Working*, *Stable*, *Consolidated*, *Disable*), extension inheritance [Moro et al 2001a, 2001b, 2001c, 2001d].

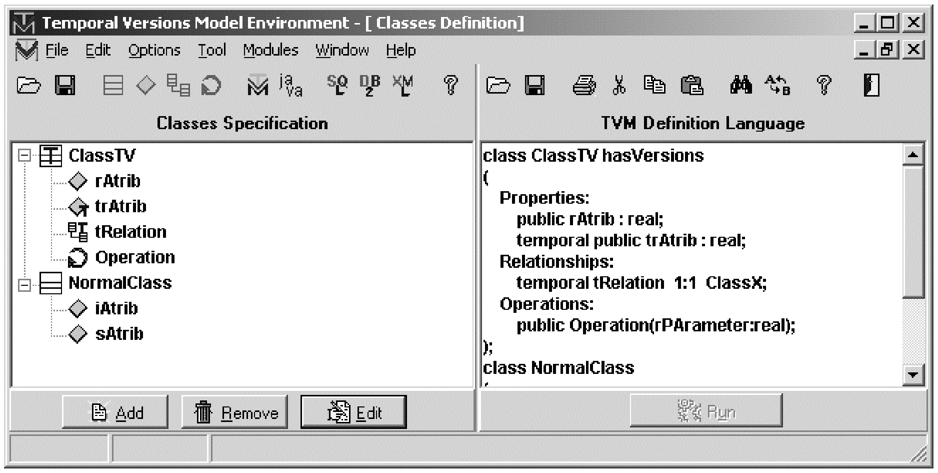
Reference [Moro et al 2001b] presents a complete modeling example and a comparative study with models OODAPLEX [Wuu and Dayal 1993] and TVOO [Rodriguez, Ogata and Yano 1999]. In summary, TVM proposes a hierarchy as the basis for the user classes specifications, while TVOO uses a formal framework, and the OODAPLEX extension works with a hierarchy of abstract time types. TVM and OODAPLEX allow temporal versioned objects among normal objects, while TVOO specifies that all objects are temporal. TVM and OODAPLEX have both transaction and valid times, while TVOO has only transaction time. Concerning versioning, each change generates a new version in TVOO, while the user must create explicitly new versions in TVM. This latter feature is important in order to limit the data space and to provide a better performance. Finally, the three models have logical exclusion and excluded objects cannot be recreated in TVOO.

3. TVM Environment

The use of a powerful temporal data model does not require a specific database management system (DBMS) that supports it to implement an application. Existing commercial DBMSs can be used for this purpose as long as a proper mapping from the temporal data model to the data model underlying the adopted DBMS is provided

[Edelweiss et al 2000]. In order to test the feasibility of this approach, TVM was mapped to a commercial database. The DB2 database was chosen because it supports of temporal data types similar to the SQL-92 standard [Snodgrass 2000]. The mapping is divided in two parts: the representation of the model's hierarchy, and the mapping of the application classes [Moro et al 2001a].

The TVM Environment has a set of interfaces for modeling classes, handling and querying instances. Specifically, in the interface for TVM class definition, the user may specify the classes in a definition tree and map the specification to TVM Definition Language and to SQL, in order to store the system in the underlying database. Figure 3 presents the interface and part of a simple example.



```
class ClassTV hasVersions
(
  Properties:
    public rAtrib : real;
    temporal public trAtrib : real;
  Relationships:
    temporal tRelation  1:1  ClassX;
  Operations:
    public Operation(rPArameter:real);
);
class NormalClass
```

Figure 3. TVM Environment, classes specification and TVM definition language

4. Summary and Concluding Remarks

This paper proposes the Temporal Versions Model (TVM) to be used as a data modeling alternative. TVM is ideal for modeling time-evolving systems that need to manage design alternatives as versions. Among its main features are: integration with existing specifications, storage of evolution and design alternatives, branched time order (not usual in the models found on the literature in which the time order is almost always linear), homogeneous treatment for valid and transaction timestamps.

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; operations for generating configurations, restoring logically removed objects, and creating an entity based on an existing entity.

This work is part of a project that aims to implementing an integrated environment software 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 [Moro et al 2001a, b, c, d]. TVM has a query language already defined [Moro et al 2001e], [Moro et al 2002]. 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 proposed.

References

Chien, S-Y., Tsotras, V.J., and Zaniolo C. (2001) “Efficient Management of Multiversion Documents by Object Referencing”, In: Procs. of Intl. Conf. of Very Large Data Bases - VLDB, Italy. p. 291-300.

Conradi, R. and Westfechtel, B. (1998) “Version Models for Software Configuration Management”, ACM Computing Surveys, vol. 30, no. 2, June 1998. p.232-282.

Edelweiss, N., Hübler, P., Moro, M.M. and Demartini, G. (2000) “A Temporal Database Management System Implemented on Top of a Conventional Database”, In: Procs. of Intl. Conf. of the Chilean Computer Science Society, Chile. p. 58-67.

Etzion, O., Jajodia, S. and Sripada, E. (1998) “Temporal Databases: Research and Practice”. LNCS 1300. Springer-Verlag.

Golendziner, L.G., Santos, C.S. dos (1995) “Versions and configurations in object-oriented database systems: a uniform treatment”, In: Procs. of Intl. Conf. on Management of Data, India, 1995, pp. 18-37.

Moro, M.M., Saggiorato, S.M., Edelweiss, N. and Santos, C.S dos (2001a) “Adding Time to an Object-Oriented Versions Model”, In: Procs. of Intl. Conf. on Database and Expert Systems Applications - DEXA, Germany, LNCS 2113, pp. 805-814.

Moro, M.M., Saggiorato, S.M., Edelweiss, N. and Santos, C.S dos (2001b) “Dynamic Systems Specification using Versions and Time”, In: Procs. of Intl. Database Engineering and Applications Symposium - IDEAS, France, IEEE Press, pp. 99-107.

Moro, M.M., Saggiorato, S.M., Edelweiss, N. and Santos, C.S dos (2001c) “A Temporal Versions Model for Time-Evolving Systems Specification”, In: Procs of Intl. Conf. on Software Engineering & Knowledge Engineering - SEKE, Argentina. pp.252-259.

Moro, M.M., Saggiorato, S.M., Edelweiss, N. and Santos, C.S dos (2001d) “Um Modelo Temporal de Versões para Especificação de Aplicações”, In: Iberoamerican Work. on Requirements Engineering and Software Environments, p.336 - 347.

Moro, M.M., Gelatti, P.C., Gomes, C.H.P., Rossetti, L.L.F., Zaupa, A.P., Edelweiss, N., Santos, C.S. dos (2001e) “Linguagem de Consultas para o Modelo Temporal de Versões”, Research Report, R.P. 308, Porto Alegre: UFRGS.

Moro, M.M., Zaupa, A.P., Edelweiss, N. and Santos, C.S dos (2002) “TVQL – Temporal Versioned Query Language”, In: 13th International Conference on Database and Expert Systems Applications - DEXA, to be published, France.

Rodríguez, L., Ogata, H. and Yano, Y. (1999) “TVOO: A Temporal Versioned Object-Oriented data model”, In: Information Sciences, Elsevier, 114. p. 281-300.

Snodgrass, R.T. (2000) “Developing Time-Oriented Database Applications in SQL”. Morgan Kaufmann Publishers.

Tansel, C.G. et al. (1993) “Temporal Databases - Theory, Design and Implementation”. Benjamin/Cummings.

Wuu, G.T.J and Dayal, U. (1993) “A Uniform Model for Temporal and Versioned Object-Oriented Databases”, In: Temporal Databases: Theory, Design, and Implementation, Edited by A.Tansel et al, Benjamin/Cummings, p. 230-247.