Evolution Specification of Multigranular Temporal Objects

Elena Camossi[1] Elisa Bertino[1] Giovanna Guerrini[2] Marco Mesiti[3]

[1]DSI - Università di Milano - Italy
{bertino, camossi}@dsi.unimi.it

[2]DI - Università di Pisa - Italy
guerrini@di.unipi.it

[3]DISI - Università di Genova - Italy
mesiti@disi.unige.it

Abstract

The main key feature of temporal databases is to maintain all values taken by object attributes over time. Since historical information may be needed at different levels of detail, multigranular temporal databases have been introduced, in which different attributes can be stored at different temporal granularities. An important issue that has not been addressed, however, is that the required level of detail does not only depend on attribute semantics, rather it is often related to how recent data are. It is quite natural that recent data are needed at greater level of detail, whereas less detail is needed as data age. As an extreme case, data can also expire, that is, they are no longer needed after a certain period of time. In this paper we address the problem of evolution and expiration of historical data in a multigranular temporal object model.

1. Introduction

Temporal databases are becoming more and more relevant in several application areas, such as meteorological forecasts, banking analysis, checking of tax contributions. The possibility to store all values an attribute takes over time allows one to compute aggregate functions on these values and to identify trends on future values. For example, in a meteorological database, the daily temperature of a city for the last year can be aggregated at season level in order to determine the season average temperature. Moreover, such information, together with the same information for the previous ten years, can be used to forecast the temperature of the next year. Multigranular temporal databases enhance the expressive power of temporal databases allowing one to handle temporal attributes at different levels of detail. Thus, in the meteorological database example, it is possible to store, for a given city, temperature at daily level, pollution at hourly level, and wind intensity and direction at each minute.

A main drawback of such databases, however, is that the amount of values that can be stored into the database and accessed when processing queries increases faster than in traditional databases. Moreover, data acquired at a fine level of detail are useful when they are acquired, but they become less important after a period of time. Very often, after a period of time only aggregate data are relevant. For example, details on the kernel temperature in a nuclear power station are relevant at granularity of seconds when they are acquired, but after a month only the daily average is relevant. Detail values could be removed from the database, for freeing storage space, or moved to tertiary storage devices. In the first case only the aggregate values can be accessed by database users, whereas, in the second case, the detail data are still accessible, but query processing time increases.

The previous discussion points out two requirements that arise in multigranular temporal databases. The first one is the possibility of aggregating temporal values at different levels of details. This aggregation, called *attribute evolution*, should be materialized in order to facilitate answering queries that require the aggregation of such temporal values. The second one is the possibility of deleting/moving old values at a certain level of detail, in order to minimize the disk storage space (we handle moving of data to tertiary storage as deletion, then we will simply refer to deletion in what follows). Such two mechanisms are orthogonal. That is, for a certain attribute both the evolution from a granularity to another and the deletion of values at a certain granularity can be specified, while for another attribute the attribute values at a certain granularity can be deleted without having to aggregate them at a coarser granularity, or they can be aggregated at a coarser level without deleting them.

To address the above requirements, in this paper we present a multigranular temporal object oriented data model in which *expiration* of attribute values at a certain granularity can be specified, together with the action to take when data expire: either aggregation at a coarser granularity, or deletion, or both. Thus, in our model, at schema level, it is possible to specify that an attribute can be evolved to a coarser granularity after a period of time. The way in which the attribute evolves is specified by means of *coercion functions* [13, 15]. Such an approach allows one to obtain summarized information – aggregates, selections or user defined – on historical data. The model supports as well the possibility of specifying the deletion of values corresponding to a set of granules, namely the oldest ones, from the database.

To develop such a mechanism, we need first to revise the notion of multigranular temporal object model, to allow different portions of the value of a temporal attribute to be stored at different granularities. Attribute values at different granularities are obtained through a coercion function, that depends on attribute semantics. This revised model, T_ODMGe, is obtained as an extension of the T_ODMG model defined in [15], which is, in turn, a multigranular temporal extension of ODMG object model. A language to specify attribute evolution and deletion depending on expiration periods is then proposed. We assume that the user specifies such information at schema definition level because the kind of evolution and deletion depends on the attribute semantics and on specific policies of the application domain. For instance, tax records have to be kept for 5 years, whereas details on bank transactions of an account have to be kept for 60 days. In the last example, moreover, after the 60 days only the account balance has to be maintained for the next 60 days. Finally, we discuss how the mechanism has been implemented in a prototype of the model we have developed.

Throughout the paper we will refer to an Accountant that wishes to store in a database information about tax declarations of his customers. Relying on current laws, the Accountant should maintain for at least five years the tax records of a given year. Actually, this is the period of time during which tax inspections are possible. Moreover, in order to check how the customer activities grow in longer periods of time, the Accountant wishes to store for each customer the average tax payments for the last five years. Finally, after ten years, starting from the average tax payments previously computed, he/she wishes to know the maximum payment.

The paper is organized as follows. Section 2 presents T_ODMGe, that revises the T_ODMG model to allow values of a temporal attribute to be stored at different granularities. Section 3 describes the syntax by means of which evolutions and deletions are expressed in the database schema. How the evolution and deletion operations are executed is discussed in Section 4. Section 5 discusses related work. Finally, Section 6 concludes the paper and outlines future research directions.

2. T_ODMGe: A Data Model supporting Evolution Attributes

In this section we present T_ODMGe, a multigranular temporal object-oriented data model that supports the representation of attributes whose values are tuples of temporal values. This is the key feature of our model. Referring to the running example, values for the tax payment attribute can be maintained at one year, five years and ten years granularities. The temporal value stored in the first position of the tuple is the value at the basic granularity, i.e., the more detailed granularity at which values for this attribute are inserted in the database. The other temporal values are computed starting from this value. A temporal value at position i in the tuple is obtained by applying a coercion function on the temporal value that is stored at position $i-1$. Therefore, moving from left to right in the tuple, temporal values for the same attribute are stored at coarser levels of detail. A sequence of coercion functions are specified in the schema for each attribute for which some evolutions are specified, to state how the temporal value at granularity i is obtained from the one at granularity $i-1$.

Evolution of attribute values is thus modeled by considering the same type system defined for T_ODMG. Only definitions of class, object and object consistency have been modified to include evolution and deletion notions. This allows us to increase the expressive power of the model yet retaining the underlying type system. In the reminder of the section we will focus on the structural characteristics of our model, i.e. attributes, disregarding methods and relationships. This is because the introduction of evolution attributes only affects the structural components of the model, whereas the behavioral components do not change.

The section is organized as follows. Section 2.1 briefly recalls some key features of T_ODMG which T_ODMGe inherits: granularities, type system, legal values, and coercion functions. Then, Section 2.2 deals with the new features of T_ODMGe. The definitions of class and object are revisited and constraints an object must meet in order to be a valid instance of a class are stated.

2.1. Granularities, Type System, Legal Values and Coercion Functions

We adopt the definition of temporal granularity given in [2], that is the standard definition of temporal granularity, commonly adopted by the temporal community. Intuitively, a temporal granularity is defined as a mapping from an ordered index set to the set of possible subsets of the *time domain*. Examples of commonly used granularities are: $days$, $months$, $years$. Every portion of the time domain obtained by such a mapping is called *granule* (e.g., 07/04/2002 represents a granule for the granularity $days$, 2002 represents a granule for the granularity $years$ and so on). Each non-empty granule of a granularity G is specified also by an index (e.g., $G(i)$ is the i^{th} granule of granularity G). Granules of the same granularity can not overlap each other, and they must keep the same order given by the index set. Let G be a granularity, a *temporal element* [7] Υ^G is a set of granules of granularity G. If this set is made by contiguous granules, we obtain a *temporal interval* [12], denoted by $[i, j]^G$, where i and j are indices of granules that bound the interval. The set of granularities managed by the model are related by the finer than relationship. A granularity G is said to be *finer than* a granularity H, denoted $G \preceq H$, if, for each index i, there exists an index j such that $G(i) \subseteq H(j)$ [3] (e.g., $days$ is finer than $months$). The symbol "$\prec$" will be used to denote the anti-reflexive finer than relationship.

Let $\mathcal{T}$ be a set of types, including class and literal types. For each type $\tau \in \mathcal{T}$, called inner type, and granularity $G \in \mathcal{G}$, a corresponding temporal type, $temporal_G(\tau)$, is defined. The set of temporal types is denoted by $\mathcal{TT}$. The temperature attribute of the meteorological database described above, representing the temperature in Celsius degrees taken every day in a particular city, could be of type $temporal_{days}(float)$, and a legal value for such an attribute could be {<06/11/2001, 23.5>, <06/12/2001, 24>, <06/15/2001, 22>}. This means that, in a particular city, the temperature on 11 June 2001 was of 23.5 degrees, on 12 June 2001 was of 24 degrees and so on. Note that we represented the legal value of temperature attribute as a set of pairs. Intuitively, a temporal value is defined as a set of partial functions that map subsets of the time domain on the set of legal values for the inner type of the corresponding temporal type. We refer to the set of time instants for which these partial functions are defined as the *domain* of the temporal value. In what follows, given a type $\theta \in \mathcal{T} \bigcup \mathcal{TT}$, we use the symbol $[\![\, \theta \,]\!]$ to denote the set of legal values for type θ.

Let v be a value of type $temporal_G(\tau)$, we denote with $v(i)$ the value of v in i^{th} granule of G. Since temporal values are partial functions, given a temporal value v, an index $i \in \mathcal{IS}$ may exist such that $v(i) = \perp$. We assume that, for each granularity H, such that $H \preceq G$, and for each pair of indices i, j, such that $H(j) \subseteq G(i)$, the value of v in granule j of H is the one in the i^{th} granule of G. This assumption is known in the temporal community as *downward hereditary property*. Given a temporal value v of type $temporal_H(\tau)$ and a temporal element Υ^G, such that granularities G, H are related by the finer than relationship, the *temporal value restriction* of v to Υ^G, denoted by $v_{|\Upsilon^G}$, intuitively is the portion of v that intersects the granules in Υ^G. In our model, coercion functions allow to perform evolution of temporal attributes. Coercion functions have been defined in T_ODMG to convert temporal values from a given granularity into values of a coarser granularity in a meaningful way. Let $\tau_1 = temporal_G(\tau)$ and $\tau_2 = temporal_H(\tau)$ be two temporal types such that $H \preceq G$. A *coercion function*:

$$C : [\![\, temporal_H(\tau) \,]\!] \rightarrow [\![\, temporal_G(\tau) \,]\!]$$

is a total function that maps values of type $temporal_H(\tau)$ into values of type $temporal_G(\tau)$. Coercion functions can be classified into three categories: *selective*, *aggregate*, and *user-defined* coercion functions. Selective coercion functions are `first`, `last`, `Proj(index)`, `main`, and `all`, whose meaning is quite obvious [13]. Aggregate coercion functions are `min`, `max`, `avg`, and `sum` corresponding to the well-known SQL aggregate functions. User-defined coercion functions correspond to methods declared in some class of the database schema. In computing aggregate coercion functions we consider undefined values (i.e., $v(i) = \perp$ for such i) as *null* values in OQL.

2.2 Classes, Objects, and Object Consistency

A T_ODMGe class declaration allows the definition of the external specification of an object type. A class specification is an object type external specification according to ODMG terminology. A class declaration consists in a class identifier, that represents the object type of the class, and a set of attributes. Each attibute has a name and a type. As in T_ODMG, an attribute of a T_ODMGe class can be temporal, if we are interested in storing values it has taken over time, or static, if only the current value of the attribute is kept. Temporal attributes have a temporal type at a certain granularity as domain, whereas static attributes have a static type as domain. To allow the storage of different portions of the history of a temporal attribute at different granularities, the domain of evolution attributes is a Cartesian product of temporal types, each with the same inner type and with a different granularity. Since, however, different components of this Cartesian product represent the same temporal information but expressed at different levels of detail, the attribute definition also contains a tuple of coercion functions, stating how to obtain values at a given granularity from the values of the corresponding granules at the finer granularity.

Example 1 *Referring to the tax payment running example, suppose to define a class* `taxpayer`*, that represents information about an accountant custumer and his/her tax payments. The class attributes are: fiscal_code (static attribute), address (temporal attribute) and tax_payments (evolution attribute). Then, (*`taxpayer`*, attr) is a class specification where attr* = {(`string`*, fiscal_code,* $\emptyset$*), (*$temporal_{years}$(`string`)*, address,* $\emptyset$*), (*$temporal_{years}$(`float`) $\times$ $temporal_{5years}$(`float`) $\times$ $temporal_{10years}$(`float`)*, tax_payments, (avg,max))*}. $\diamond$

A $T_$ODMGe object, as in $T_$ODMG, is defined as a 6-tuple (id,N,v,c,$[i,j]^{G_I}$) where id is the object identifier, N is the set of object names, v is the object state, that is, the attribute values, c is the most specific class to which the object belongs, and, finally, $[i,j]^{G_I}$ is a temporal interval representing the object lifespan, expressed at the chronon granularity, that is the granularity of the time domain. Differently from $T_$ODMG, the state of an object can contain, for some attributes, values that are tuples of temporal values at different granularities.

Example 2 *Consider the class* `taxpayer` *of Example 1. Suppose that* $\mathtt{i}_1$ *is an object identifier and* `Smith` *is an object name. An object of class* `taxpayer` *is* $(\mathtt{i}_1, \{\mathtt{Smith}\}, v, \mathtt{taxpayer}, [1,\infty]^{G_I})$*, where* v *is the object state depicted in Figure 1.* ◇

Intuitively, in ODMG an object is a consistent instance of a class if its state matches the class definition. That is, each attribute value is a legal value of the corresponding attribute type and a legal value is specified for each attribute type. In $T_$ODMG this notion has been revisited in order to consider the time dimension. Specifically, constraints have been imposed on the relationships between the lifespan of an object and the domain of the temporal attributes it contains, in that the domain of a temporal attribute should not exceed the lifespan boundary. In $T_$ODMGe the notion of object consistency has been further extended for handling the presence of evolution attributes. Intuitively, constraints that should be verified for these attributes are the following:

1. the value of an evolution attribute is a tuple of temporal values; each such value, in turn, is a legal value for the corresponding temporal type in the class definition;
2. for each value defined, the temporal interval associated with the corresponding granule intersects the object lifespan;
3. temporal values in the tuple, for temporal value restrictions corresponding to the same portion of the time domain, are related by the corresponding coercion function.

We refer the interest reader to [6] for the formal definition of $T_$ODMGe consistent instance. Note that this consistency notion requires that, if data at different granularities are available, they should be coherent, that is, for each granule, the value at coarser granularity must be the one obtained by applying the coercion function to the corresponding values at the finer granularity. By contrast this is not required if data at the finer granularity are not part of the object state anymore, because they have been deleted.

Example 3 *Consider the class* `taxpayer` *of Example 1 and the object* i_1 *of Example 2, which state* v *is represented in Figure 1. The attribute* `tax_payments` *is an evolution attribute, and its value is a triple of temporal values. For this attribute, disregarding value deletions marked with crosses, we could check that the temporal value at* $5years$ *granularity has been obtained applying coercion function* avg *to the temporal value at granularity* $years$*. In the same way, the temporal value at* $10years$ *granularity has been obtained applying coercion function* max *to the temporal value at granularity* $5years$*.* ◇

3 Specification of Evolution and Deletion of Temporal Attributes

Conditions for attribute evolution and deletion often depend on attribute semantics and are a-priori known, therefore they can be specified at schema definition time. For this reason the $T_$ODMG class definition language has been extended in order to allow a user to declaratively specify evolution attributes. The temporal type of an evolution attribute is specified as for a normal temporal attribute. Additionally, for such an attribute it is possible to specify: the various granularities at which the temporal values can be stored and, for each representation level, the coercion function to convert the attribute value to the immediately coarser level; the possible deletion operation; the frequencies at which evolution and deletion should be performed. Since evolution and deletion are orthogonal operations, they are specified separately by means of `evolve` and `delete` clauses that will be discussed in detail in the remainder of the section.

Figure 2 shows the BNF grammar to use in order to declare an evolution attribute. In the figure, terminal symbol *attr_name* denotes an attribute name; *g_name* denotes a granularity name; *num* denotes a natural number greater than zero; *c_function* denotes a coercion function name; *static_type_name* denotes a type name for a static type. In the remainder of the section, Section 3.1 presents the language to specify evolutions. Section 3.2 describes the language for the deletion specification, whereas Section 3.3 slightly describes the checks performed by the specification language interpreter.

3.1 Specification of Evolutions

Attribute evolutions are specified by means of `evolve` clauses. An evolve clause states the target granularity of an evolution operation, the frequency at which the evolution should be performed and the coercion function to apply. The user may specify more than one `evolve` clause for an evolution attribute, each with a different target granularity. The number of evolution clauses declared for an attribute corresponds to the number of representations, one for each target granularity, at which the user wishes to represent the attribute. Coercion functions that may appear in an

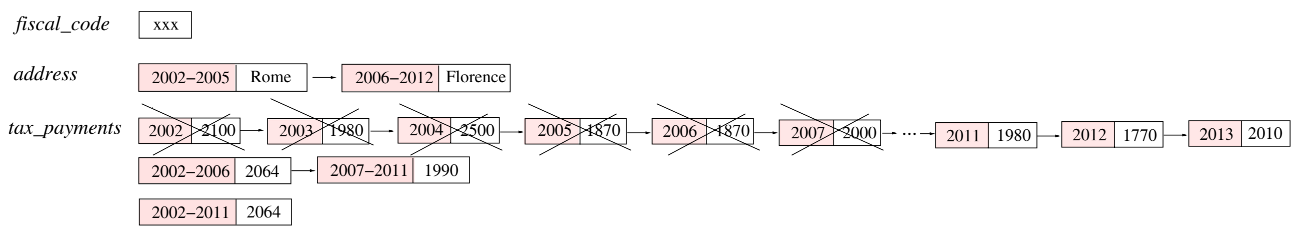


Figure 1. Example of object state

```
⟨attr_decl⟩       ::=  attribute ⟨type⟩ attr_name; |
                       attribute ⟨temporal_type⟩ attr_name {⟨evolution⟩};
⟨evolution⟩       ::=  [⟨e_list⟩] [⟨delete⟩;]
⟨e_list⟩          ::=  ⟨evolve⟩; [⟨delete⟩;]| ⟨e_list⟩ ⟨e_list⟩
⟨evolve⟩          ::=  evolve to g_name after num g_name  using c_function
⟨delete⟩          ::=  delete num g_name from g_name after num g_name
⟨type⟩            ::=  static_type_name | ⟨temporal_type⟩
⟨temporal_type⟩   ::=  temporal g_name(static_type_name)
```

Figure 2. BNF of attribute declaration

evolve clause are those presented in Section 2.1. If more than one evolution clause is specified for the same attribute, the temporal value specified at a certain granularity is used to establish the value at the immediately coarser granularity.

Example 4 *Consider class specification of Example 1. Attribute* tax_payments *represents payments of taxes of an accountant customer and is defined at* $years$ *granularity. After five years, we would know the average tax payment of previous years, and the same for following years. After ten years we wish to know the maximum average amount stored, referred to previous ten years. Then, the following declaration may be specified:*

```
attribute temporal_years(float) tax_payments {
  evolve to 5years after 5 years using avg;
  evolve to 10years after 10 years using max;
};
```
◇

In the language there are no constraints about the order of the evolve clauses. This is because the interpreter can determine their order relying on the $\prec$ relationship that exists among the different granularities.

3.2 Specification of Deletions

The deletion of values of an evolution attribute is specified by means of delete clauses. Each delete clause specifies the temporal value and the frequency at which the deletion should be performed. The frequency specification is analogous to the one for evolution. The values removed from the database are the oldest that appear in the temporal value.

Example 5 *Consider Example 4 and suppose that after five years we wish to remove the oldest values corresponding to one year of tax payments from the temporal value at* $years$ *granularity. Then, the following declaration should be added to the* evolve *clauses presented in Example 4:*

```
delete 1 years from years after 5 years;
```
◇

Deletions can be specified both for the temporal value at basic granularity (as in the previous example) and also for the other temporal values that represent the attribute evolutions. However, at most a deletion specification can be declared for each temporal value. The order of delete clauses is not relevant as for evolve clauses, because each delete clause specifies the temporal value from which values should be deleted through the corresponding granularity name, that is unique for every evolution attribute.

3.3 Temporal Consistency Checking

An user may specify, both for evolve and delete clauses, frequencies that do not make sense. Suppose, for example, to specify, for an attribute defined at granularity $days$, the evolve clause:

```
evolve to months after 1 days using max;
```

Such a specification is not correct, because we have not enough data, after one day, to perform an evolution that results in a monthly value. The T_ODMGe interpreter is able to establish if a frequency specification is right. By contrast, there are situations in which it is not possible to detect with certainty that the specification is wrong. For example, it is not possible to detect that the clause:

```
delete 5 days from days after 150 hours;
```

is wrong, because, according to the notion of granularity we consider, it is not possible to check if "150 $hours$" defines a temporal interval greater than "5 $days$". We refer to

[6] for details about the checks performed by the interpreter component of T_ODMGe.

4 Evolution and Deletion Execution

In this section we present the mechanism supporting evolution and deletion operations. This mechanism performs the evolution or the deletion of values from an evolution attribute, upon insertion of a value at the basic granularity of that attribute into the database. This operation is performed whenever the elapsed time between the validity time of the value and the validity time of the first value that has not been aggregated yet (this time is called *starting time*) is greater than the specified frequency.

In the remainder of the section, we first explain the concept of *starting time* and restrictions on the type of operations that can be performed on evolution attributes (Section 4.1). Then, we present different plausible interpretations of the frequency declared in the evolution and deletion clauses and the one we adopted (Section 4.2). Finally, we discuss the execution process of these operations (Section 4.3).

4.1 Starting Time and Update Monotony

Values of an evolution attribute are always inserted at the finest granularity among the granularities at which the attribute value is maintained. When a value for an evolution attribute is inserted into the database, in order to establish if some evolutions or deletions should be performed, the system considers, for each representation level of an attribute, the *starting time*. For each level, the starting time is the validity time of the oldest value of the level, if no evolution and deletion have been performed yet. By contrast, if such an operation has been already performed, the starting time of the level is the validity time of the first value that has not participated yet to the operation.

In our approach non-monotonic updates are allowed only for recent values of an attribute, i.e. the values whose validity time is greater than the starting time of the attribute. In this way, corrective retroactive updates for recent values are possible. By contrast, updates on values whose validity time is older than the starting time of the attribute are denied. This restriction, called *updates monotony restriction*, reminds approaches taken by transaction time models and data warehousing approaches. The difficulty of allowing insertions and updates before the starting time is on the propagation of the effects of these operations on corresponding values at coarser granularities. We are currently investigating how to further weakening the restriction.

4.2 Frequency of Evolution and Deletion

The frequency specified in `evolve` and `delete` clauses can be interpreted in two ways. Suppose to specify an evolution of an attribute whose values are inserted daily to granularity $months$, with a frequency of "6 $months$". The first interpretation of the temporal interval "6 $months$", adopted in [18, 19], is "the number of months since the first data has been stored in the database". For example, if the first attribute value has been stored on January 15^{th}, relying on such interpretation, 6 months later means July 15^{th}. By contrast, another interpretation of "6 $months$" can be "the number of months from the beginning of the month on which a specific day falls". With respect to this interpretation 6 months later than January 15^{th} means July the 1^{st}. T_ODMGe model adopts the latter interpretation, because it is more consistent with the standard notion of granularity we refer to.

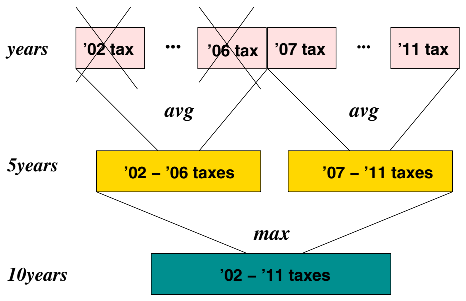
4.3 Evolution and Deletion Process

When a value of an evolution attribute is inserted and the condition for evolution or deletion holds, the evolution (or deletion) process is performed. Evolution operations are performed before the deletion ones. Both operations are performed starting from the temporal value at finest granularity moving up to the coarser ones. Each operation refers to a specific temporal value v. For each temporal value v, the system considers the temporal element, named Υ^{Tg}, at the evolution target granularity. This temporal element is the temporal element fully contained into the temporal interval bounded by the starting time of v, and the validity time of the inserted value. If this temporal element is not empty, the operation is performed. If the operation is an evolution, the system applies the coercion function specified in the database schema on the temporal value $v|_{\Upsilon^{Tg}}$, that is the temporal restriction of v to the temporal element Υ^{Tg}. The obtained value is inserted into the temporal value at the evolution target granularity Tg, that is the temporal value at immediately coarser granularity with respect to v. The same process is recursively applied to the temporal value at the immediately coarser granularity, and so on, for every evolution defined for the attribute.

Figure 3(a) shows the class specification for the tax payment running example, whereas Figure 3(b) is a scketch of what happens to the `tax_payments` attribute value on a period of ten years. In Figure 3(b) we consider a subset of the object state depicted in Figure 1. Note that after ten years since the first value for `tax_payments` attribute is inserted, two evolutions have been performed and five values at granularity years have been deleted. Note, moreover, that the value at granularity $5years$ has been kept even if all the values for the finer granularity have been removed from

```
class taxpayer {
 attribute temporal_years(string) address;
 attribute string fiscal_code;
 attribute temporal_years(float) tax_payments {
  evolve to 5years after 5 years using avg;
  delete 1 years from years after 5 years;
  evolve to 10years after 10 years using max;
 }; };
```

(a)



(b)

Figure 3. (a) Class declaration and (b) evolution and deletion schedule

the database.

5. Related Work

In this paper we address the problem of evolving temporal data at coarser levels of granularity and of deleting expired temporal data. Conditions for data evolution and deletion are based on data age and specified in the schema, as they depend on data semantics. These two issues have never been addressed together. Thus, in what follows, we briefly review the work that influenced our approach, for what concerns data evolution and data expiration, separately. We remark, however, that our model differs from other approaches we refer to because these models, except for [19], consider transaction time instead of validity time and because we work on an object-oriented data model rather than on the relational model.

5.1. Evolution

The issue we deal with in this paper has some similarities with the work developed in the area of data warehousing for materialized and persistent views [4, 9, 10, 16]. An important difference, however, is that those approaches only allow one level of aggregation: relying on the granularity of detailed values, only one level of aggregation is specified, i.e. a view is materialized on basic relations. By contrast, our approach allows the evolution of object attributes at increasing coarser levels of detail. Each evolution is computed applying a coercion function on an attribute temporal value. The obtainedc result is a temporal value at a coarser granularity and can be used by another coercion function to define another value at a coarser level of detail. Note, moreover, that we do not consider only aggregation when moving to coarser levels, rather also selective or user-defined coercion functions can be used.

The most relevant difference, however, is that in the data warehousing approaches, aggregation is applied to the whole set of values, disregarding whether they are recent or not. The age of data are, by contrast, explicitly considered in [18, 19], that, in turn, rely on the Chronicle Model [11]. In those approaches the concept of materialized *temporal views* is introduced. Relying on such concept updates on source data are automatically propagated to materialized views. Moreover, temporal windows that slide as time advances and update the materialized views are considered. Materialized views are specified as queries that can involve an arbitrary number of relations and attributes and determine aggregated values. In our model, by contrast, we evolve at a coarser level the values of a single attribute, thus, we have different temporal windows, each one local to the attribute it refers to. Moreover, our model supports different time granularities and multiple levels of aggregation, while in [18, 19] a single time granularity and only one level of aggregation are considered. Finally, as we discussed in detail in Section 4, our notion of time expiration does not refer to a specified amount of time (e.g., 24 hours for a day), rather to the end of a granule of the specified granularity. Thus, when we say that detail data are kept for one day, this means that they are aggregated when first data referring to a new day are inserted. With the sliding window approach of [18, 19] this means, by contrast, that, in the mid of a day, the detail values of the last 24 hours are kept.

5.2 Deletion

In [8] the concept of *data expiration* is introduced. Relying on such concept data can be removed (i.e., they *expire*) from the database without affecting related views. A similar approach has been proposed for the time dimension in [17]. In [17] a technique is presented supporting automatic data expiration in a historical data warehouse and preserving, at the same time, answers to a known and fixed set of first-order queries. Also in this case, only deletions from detail data are considered. This approach assumes that conditions for data expiration are not declared in the schema, rather they are deduced from a given set of queries. Such an approach is adequate if no information are known at schema definition time. However, it is not exclusive with respect to our approach, rather it can complement ours, since conditions for data expiration can be deduced for those attributes for which they are not known at schema definition time.

6 Conclusions and Future Work

In this paper we have proposed an approach to evolve data at coarser granularities, and to delete or move to tertiary storage devices expired data, based on data expiration declarations specified in the database schema. This approach has been implemented in the prototype implementation of T_ODMGe [5] we realized in ObjectStore PSEPro for Java.

We are currently extending the work presented in the paper along different directions. A first one is the possibility to allow different aggregations for the same data at the same granularity, that can also be used to optimize the computation of further aggregations at coarser levels. More general evolution mechanisms are also under consideration. We are also interested in the integration of our approach, based on explicit declaration of evolution and deletion in the schema, with the derivation of data evolution and deletion conditions based on the typical data usage. Such derived conditions could be used for those data for which no knowledge about data expiration is known at schema definition time. Another important direction we are working on is how to weaken the restriction on updates, in order to allow all retroactive updates. Finally, we are also investigating the problem of query evaluation in our multigranular model, as an extension to the query evaluation mechanisms proposed for T_ODMG in [1, 14].

Acknowledgements. The authors wish to thank Riccardo Ponta that, as part of his master thesis, worked on the model described in this paper.

References

[1] E. Bertino, E. Ferrari, G. Guerrini, and I. Merlo. Navigating Through Multiple Temporal Granularity Objects. In *Proc. of the Eigth International Symposium on Temporal Representation and Reasoning*, pages 147–155, July 2001.

[2] C. Bettini, C. Dyreson, W. Evans, R. Snodgrass, and X. Wang. A Glossary of Time Granularity Concepts. In *Proc. of Temporal Databases: Research and Practice (TIME99)*, number 1399 in LNCS, pages 406–413. Springer-Verlag, 1998.

[3] C. Bettini, S. Jajodia, and X. Wang. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer-Verlag, 2000.

[4] J. Blakeley, P. Larson, and F. Tompa. Efficiently Updating Materialized Views. In *Proc. of SIGMOD Conference*, pages 61–71, 1986.

[5] E. Camossi, E. Bertino, G. Guerrini, and M. Mesiti. A Multigranular Object Model Supporting Evolution and Expiration of Historical Data. In preparation.

[6] E. Camossi, E. Bertino, G. Guerrini, and M. Mesiti. Evolution Specification of Multigranular Temporal Objects (Extended Version). Technical report, Universit`a di Genova, 2002.

[7] S. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions On Database Systems*, 14:418–448, 1988.

[8] H. Garcia-Molina, W. Labio, and J. Yang. Expiring Data in a Warehouse. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 500–511, 1998.

[9] A. Gupta and I. Mumick. Maintenance of Materialized Views: Problems, Techniques and Applications. *IEEE Quarterly Bulletin on Data Engineering; Special Issue on Materialized Views and Data Warehousing*, 18(2):3–18, 1995.

[10] N. Huyn. Efficient View Self-Maintenance). Technical report, 1996.

[11] H. Jagadish, I. Mumick, and A. Silberschatz. View Maintenance Issues for the Chronicle Data Model. In *Proc. of Symposium on Principles of Database Systems*, pages 113–124, 1995.

[12] C. Jensen and C. Dyreson. The Consensus Glossary of Temporal Database Concepts. In *Proc. of Temporal Databases: Research and Practice*, number 1399 in LNCS, pages 366–405. Springer-Verlag, 1998.

[13] I. Merlo. *Extending the ODMG Object Model with Temporal and Active Capabilities*. PhD thesis, Universit`a di Genova, February 2001.

[14] I. Merlo, E. Bertino, E. Ferrari, S. Gadia, and G. Guerrini. Querying Multiple Temporal Granularity Data. In *Proc. of the Seventh International Workshop on Temporal Representation and Reasoning*, pages 103–114. S. Goodwin and A. Trudel, July 2000.

[15] I. Merlo, E. Bertino, E. Ferrari, and G. Guerrini. A Temporal Object-Oriented Data Model with Multiple Granularities. In *Proc. of Sixth International Workshop on Temporal Representatation and Reasoning*, pages 73–81. C. Dixon and M. Fisher, May 1999. Orlando, Florida.

[16] D. Srivastava, S. Dar, H. Jagadish, and A. Levy. Answering Queries with Aggregation Using Views. In *The VLDB Journal*, pages 318–329, 1996.

[17] D. Toman. Expiration of Historical Databases (Extended Abstract). In *Proc. The Eighth International Symposium on Temporal Representation and Reasoning (TIME'01)*, June 2001. Cividale del Friuli, Udine, Italy.

[18] J. Yang and J. Widom. Maintaining Temporal Views over Non-Temporal Information Sources for Data Warehousing. In *Extending Database Technology*, pages 389–403, 1998.

[19] J. Yang and J. Widom. Temporal View Self-Maintenance. In *Extending Database Technology*, pages 395–412, 2000.