Handling Expiration of Multigranular Temporal Objects

ELENA CAMOSSI, ELISA BERTINO and MARCO MESITI, *DICO – Università degli Studi di Milano. Via Comelico, 39/41 - 20135 Milano, Italy.*
E-mail: {bertino, camossi, mesiti}@dico.unimi.it

GIOVANNA GUERRINI, *Università di Pisa - Italy.*
E-mail: guerrini@di.unipi.it

Abstract

A well-known problem of temporal databases is that the amount of stored data tends to increase very fast. Moreover, detailed data are useful when they are acquired but they often become less relevant after some time. In most cases, after a period of time only summarized data need to be kept, whereas detailed data expire and can be removed from the database. Multigranular temporal databases enhance the expressive power of temporal databases by supporting temporal attributes at different levels of detail. However, in existing approaches the level of detail of an attribute, that is its granularity, depends only on the attribute semantics and does not depend on how recent the attribute values are. This paper proposes an approach supporting the aggregation of different portions of the value of a temporal attribute at different levels of detail, and the deletion or the transfer to tertiary storage of old values at a given level of detail, in order to minimize disk storage occupancy. In the proposed multigranular temporal object-oriented data model, the *expiration* of attribute values at a given granularity can be specified, together with the action to take when data expire: either aggregation to a coarser granularity, or deletion of values, or both.

Keywords: Temporal databases, multiple granularities, granularity evolution, deletion of temporal values, expiration of dynamic attributes.

1 Introduction

Temporal databases are becoming increasingly relevant in several application areas, such as meteorological forecasts, financial analysis, tax payment control, and medical information systems. The possibility to consider temporal or historical attributes, that is to store all values that an attribute takes over time, allows one to compute aggregate functions on these values and to identify trends of future values. For example, in a meteorological database, values of the daily temperature of a city for the last year can be aggregated at season level in order to determine the season average temperature. Such information, together with the same information for the previous ten years, can be used to predict the temperature of the next year. However, a main drawback of temporal databases is that the amount of values that need to be stored and accessed when processing queries increases faster than in traditional databases. Moreover, data at a fine level of detail are useful when they are acquired, but they often become less relevant after some time. In most cases, after a period of time only aggregate data are of interest to applications and users. For example, details on the kernel temperature in a nuclear power station are relevant at a granularity of seconds when they are acquired, but after a month only the daily average can be of interest. Detailed data can be removed from the database, in order to make available storage space, or moved to tertiary storage. In the first case only aggregate data remain available to applications and users, whereas in the second case, detailed data are still accessible, but query processing time increases.

Multigranular temporal databases enhance the expressive power of temporal databases by handling temporal attributes at different levels of detail. Thus, in the meteorological database example, it is possible to store, for a given city, the temperature for each day, the pollution for each hour, and the wind intensity and direction for each minute. However, it is important to point out that the required level of detail does not depend only on the attribute semantics, rather it is often related to how recent data are. As discussed above, it is quite natural that recent data be provided with a greater level of detail, whereas less detail is needed as data age. Despite its practical importance, data models and systems for multigranular temporal databases have not addressed such a requirement so far.

Even though one may come up with a variety of multigranular temporal data models and systems able to relate detail levels with data age, we believe that there are two crucial functionalities that should always be provided. The first one deals with capabilities for computing and materializing aggregates of temporal values at different levels of detail. Materialization, in particular, is instrumental for efficiently answering queries that require aggregated temporal values. The second one deals with capabilities for deleting/moving out of the online database past values at a given level of detail, in order to minimize the disk storage space.[1] Mechanisms implementing these two capabilities are orthogonal. That is, for a given attribute both the evolution from a granularity to another one and the deletion of values at a certain granularity can be specified, while for another attribute its 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 granularity without deleting them.

In this paper we address the above requirements by presenting T_ODMGe, a multigranular temporal object-oriented data model which supports the specification of *dynamic attributes*. A dynamic attribute is a temporal attribute for which *expiration conditions* are specified. An expiration condition is given specifying an *expiration frequency* for the attribute value at a certain granularity, and the action to take when data expire: either evolution at a coarser granularity, or deletion of values, or both. If the action specified is a deletion, the expiration condition must also specify the amount of data to expire, expressed through a *temporal period*.

Thus, in our model, at schema level, it is possible to specify that an attribute granularity can be evolved to a coarser granularity after a period of time. The way in which the attribute granularity evolves is specified by means of *coercion functions* [2]. Such an approach allows one to obtain summary information – through aggregation, selection, or user defined operations – from historical data. The model also supports the possibility of specifying the deletion of values corresponding to a set of granules, namely the oldest ones, from the database. To develop such an expiration mechanism, we need first to revise the notion of multigranular temporal object model, so that different portions of the value of a temporal attribute can be stored at different granularities. Attribute values at different granularities are related by means of coercion functions. The coercion functions applied depend on attribute semantics. T_ODMGe model is obtained as an extension of the T_ODMG model defined in [2], which is, in turn, a multigranular temporal extension of the ODMG object model [9].

A language to specify attribute granularity evolution and value deletion is then proposed. We assume that the user specifies such information at schema definition level because expiration conditions depend on the attribute semantics and on specific policies of the application domain. For instance, according to current Italian laws, 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 60 days, only the account balance has to be maintained for the next 60 days. We then investigate how expiration frequencies and amount of temporal data to delete, both expressed as temporal periods, can be interpreted in a temporal model based on valid time like ours, and we propose a lazy ap-

[1] We handle moving of data to tertiary storage as deletion, then we will simply refer to deletion in what follows.

proach to dynamic attribute management. The semantics of dynamic attributes is then investigated, discussing how update operations on these attributes are performed, and how they can trigger the execution of expiration operations. Particular attention is devoted to the management of nonmonotonic updates and to the incremental re-computation of aggregations in case of updates to portions of an attribute that have already participated in an expiration.

Running example: tax declarations. Throughout the paper we will refer to an Accountant who wishes to store in a database information about tax declarations of his customers. According to current Italian laws, the Accountant should maintain for at least five years the tax records of each given year. This is indeed the period of time during which tax inspections are possible. Moreover, in order to analyze the trends of customer activities 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 wishes to record only the maximum one.

Paper outline. The paper is organized as follows. Section 2 presents T_ODMGe, which extends the T_ODMG model to support the representation of the values of a temporal attribute at different granularities. Section 3 presents the syntax for expiration specification in a database schema. Section 4 discusses the time dimension and interpretation of temporal frequencies adopted in T_ODMGe. Section 5 deals with the execution of the expiration operations. Section 6 discusses related work, while Section 7 concludes the paper and outlines future research directions.

2 T_ODMGe: a data model supporting dynamic attributes

In this section we present T_ODMGe, a multigranular temporal object-oriented data model that supports attributes whose values are tuples of temporal values (that is historical values) expressed using different granularities. Different portions of the attribute value over time, indeed, can be stored at different levels of detail, and are thus represented by temporal values at different granularities. This is the key feature of our model. Referring to the running example, since values for the tax payment attribute can be maintained at one year, five years and ten years granularities, the tuple value for this attribute contains three temporal values, one for each granularity we consider. Note that, since we introduce in T_ODMGe the possibility to delete portions of historical values, a tax payment value referring to a particular instant is not nececessarly present into the database at all of these levels of detail, that is the three temporal values do not necessarly cover the same amount of time and in particular the same time interval.[2] In the general case, the value of an attribute represented at different levels of detail is a tuple of temporal values that refers to different time intervals that potentially intersect each other.

The components in a tuple value are ordered from the finest granularity to the coarsest one. Then, the first component of the tuple value of the running example is the temporal value at granularity year, the second component is the one at granularity five years, and the last component is the temporal value expressed at granularity decades.

The temporal value stored as the first component of a value tuple, which is the temporal value that is updated directly by the user to perform attribute updates, will be referred to as the value at the *base granularity* in the tuple for the attribute, that is the finest granularity declared for it. Moreover, the other temporal values that appear in the tuple are computed starting from this value. A temporal value stored in the ith component of the tuple is obtained by applying a coercion function to the temporal

[2] We refer to the time interval bounded by the oldest and the more recent time instants considered to timestamp a temporal value.

value stored in the $(i-1)$th component. Therefore, by considering values from finest granularity to coarser ones within the tuple, temporal values for the same attribute are stored at coarser levels of detail. A tuple of coercion functions is specified in the schema for each dynamic attribute for which evolutions are specified, to state how the temporal value at granularity i is obtained from the one at granularity $i-1$.

The expiration of attribute value is modelled by considering the same type system defined for T_ODMG. Only the definitions of class, object, and object consistency are affected by the inclusion of evolution and deletion notions. This approach allows us to enhance the expressive power of the base model, that is T_ODMG, yet retaining the underlying type system. Since the introduction of dynamic attributes affects only the structural components of the model, in the remainder of the section, we will focus on its structural characteristics, that is attributes, disregarding methods and relationships.

Since T_ODMGe, as an extension of T_ODMG, retains the basic notions of granularities, temporal types and values, and coercion functions, the presentation of the model is organized as follows. Section 2.1 briefly recalls the key features of T_ODMG which T_ODMGe inherits. Then, Section 2.2 and Section 2.3 deal with the new features of T_ODMGe. Specifically, Section 2.2 revisits the definitions of T_ODMG class and object, whereas Section 2.3 presents the constraints an object must meet in order to be a consistent instance of a class.

2.1 Granularities, temporal types and values, coercion functions

We adopt the definition of temporal granularity given in [3], that is the standard definition of temporal granularity, commonly adopted by the temporal database community. Intuitively, a temporal granularity is defined as a mapping from an ordered index set $\mathcal{IS}$ to the set of possible subsets of the *time domain*. We assume that the time domain is a discret set of time instants, on which a total order relation is defined. 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. using a textual representation for granules, '$07/04/2002$' represents a granule for the granularity $days$, '2002' represents a granule for the granularity $years$, and so on). Each nonempty granule of a granularity G can be also specified by an index (e.g. $G(i)$ is the ith granule of granularity G). Granules of the same granularity cannot overlap and must keep the same order given by the index set. Granularity G_I is the chronon granularity [16], that is the granularity of the time domain, in which each granule consists of a single time instant, corresponding to its index. Let G be a granularity, a *temporal element* [10] Υ^G is a set of granules of granularity G. If this set contains contiguous granules, we obtain a *temporal interval* [15], denoted by $[i,j]^G$, where i and j are the indices of granules that bound the interval. A temporal interval denotes a segment of the time domain. The length of such a segment is called *temporal period*. The set of granularities managed by the model is denoted by $\mathcal{G}$. Granularities in $\mathcal{G}$ are related by the finer-than relationship. A granularity G is said to be *finer-than* a granularity H, denoted by $G \preceq H$, if, for each index i, an index j exists such that $G(i) \subseteq H(j)$ [5] (e.g. $days$ is finer-than $months$). We also say that H is *coarser-than* G. The symbol '$\prec$' denotes the anti-reflexive finer-than relationship.

Let $\mathcal{T}$ be a set of types, including class and literal types denoted by $\mathcal{OT}$ and $\mathcal{LT}$, respectively. 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 $\mathcal{TT}$. A temporal value of a temporal type $temporal_G(\tau)$ is defined as a partial function that maps G-granules (referred to through their indices) to τ values. We refer to the set of time instants for which these partial functions are defined as the *domain* of the temporal value. The temperature attribute of the meteorological database

described above, representing the temperature in Celsius degrees taken every day in a particular city, could be specified of type $temporal_{days}(float)$;[3] an example of legal value for such an attribute is {'⟨11/06/2001, 23.5⟩', '⟨12/06/2001, 24⟩', '⟨15/06/2001, 22⟩'} $_{days}$. Such a value records that, in a particular city, the temperature on 11 June 2001 was 23.5 degrees, on 12 June 2001 was 24 degrees, and on 15 June 2001 was 22 degrees . Note that the legal value of the temperature attribute has been represented as a set of pairs (corresponding to the graph of the function). In what follows, given a type $\tau \in \mathcal{T} \bigcup \mathcal{TT}$, the notation $[\![\tau]\!]$ denotes the set of legal values for type τ, whereas $[\![\tau]\!]_t$ denotes the set of legal values for type τ at time instant *t* [2].

Let v be a value of type $temporal_G(\tau)$, we denote with $v(i)$ the value of v in the ith 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 G and H, such that $G \preceq H$, and for each pair of indices i, j, such that $G(i) \subseteq H(j)$, the value of v in granule i of G is the one in the jth granule of H. 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 $G \preceq H$, the *temporal value restriction* of v to Υ^G, denoted by $v_{|\Upsilon^G}$, intuitively is the portion of the domain of v that intersects the set of indexes of the granules in Υ^G.

In our model, coercion functions allow one to evolve the granularity 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 $temporal_G(\tau)$ and $temporal_H(\tau)$ be two temporal types such that $G \prec H$. A *coercion function*:

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

is a total function that maps values of type $temporal_G(\tau)$ into values of type $temporal_H(\tau)$. 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`. Coercion function `proj(index)`, for each granule in the coarser granularity, returns the value corresponding to the granule of position `index` at the finer granularity. Coercion function `first` and `last` are the obvious specializations of the previous one. Coercion function `main`, for each granule in the coarser granularity, returns the value which appears most frequently in the included granules at the finer granularity. Coercion function `all`, for each granule in the coarser granularity, returns the value which always appears in the included granules at the finer granularity if this value exists, the null value otherwise. 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 the classes of the database schema. In computing aggregate coercion functions we consider undefined values (i.e. values at granules i such that $v(i) = \perp$) as *null* values in OQL.

2.2 T_ODMGe classes and objects

A T_ODMGe class declaration allows the definition of the external specification of an object type. A class declaration consists of 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. In T_ODMGe, we can define also *dynamic attributes* to support the representation of different portions of the history of a temporal attribute according to different granularities. The domain of a dynamic attribute is a Cartesian product

[3] This means that the attribute values are maintained at the *days* granularity.

of temporal types, each with the same inner type and with a different granularity. Since each component of this Cartesian product basically represents the same temporal information, though expressed at a different level of detail, the attribute definition also contains a tuple of coercion functions, specifying how values at a given granularity are generated from the values of the corresponding granules at the finer granularity. The following definition formalizes the notion of attribute specification in T_ODMGe, which represents the main difference between the definition of classes in T_ODMGe and T_ODMG.

DEFINITION 2.1 (T_ODMGe attribute specification)
The attribute specification $attr$ of a class c is a set containing an element for each attribute of the class. Each element is a triple $(a_{type}, a_{name}, a_{coerc})$, where:

a_{type} is the attribute domain type, i.e.:

- $a_{type} \in \mathcal{LT} \cup \mathcal{OT}$, or
- $a_{type} = (\tau_1, \ldots, \tau_n),\ such\ that\ n \geq 1,\ \tau_j = temporal_{G_j}(\tau), 1 \leq j \leq n,\ \tau \in \mathcal{LT} \cup \mathcal{OT},\ G_1 \prec \ldots \prec G_n$;

a_{name} is the attribute name;

a_{coerc} is a tuple of coercion functions, such that:

$$a_{coerc} = \begin{cases} () & \text{if } a_{type} \in \mathcal{LT} \cup \mathcal{OT} \\ (C_1, \ldots, C_{n-1}) & \text{if } a_{type} = (\tau_1, \ldots, \tau_n) \end{cases}$$

where, for each for each $i,\ 1 \leq i \leq n-1,\ C_i : [\![\tau_i]\!] \rightarrow [\![\tau_{i+1}]\!]$.

EXAMPLE 2.2
Consider the tax payment example. Suppose that a class `taxpayer` is specified representing information about customers and their tax payments. The class attributes are: fiscal_code (static attribute), address (temporal attribute), and tax_payments (dynamic attribute). Then, (`taxpayer`, $attr$) is a class specification where:

$attr$ = {(`string`, fiscal_code, ()), ($temporal_{years}$(`string`), address, ()), ($temporal_{years}$(`float`) $\times$ $temporal_{5years}$(`float`) $\times$ $temporal_{decades}$(`float`), tax_payments, (avg,max))}.

A T_ODMGe object, as a T_ODMG object, is defined as a 5-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 (i.e. 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. Unlike in T_ODMG, the state of an object can contain, for some attributes, values that are tuples of temporal values at different granularities as stated in the following definition.

DEFINITION 2.3 (T_ODMGe object state)
Given an object o, its state v is a tuple $(a_1 : v_1, \ldots, a_n : v_n)$, where each v_i, $1 \leq i \leq n$, is a static value or a tuple of temporal values.

EXAMPLE 2.4
Consider the class `taxpayer` of Example 2.2. Let $\mathtt{i}_1$ be an object identifier and `Smith` 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.

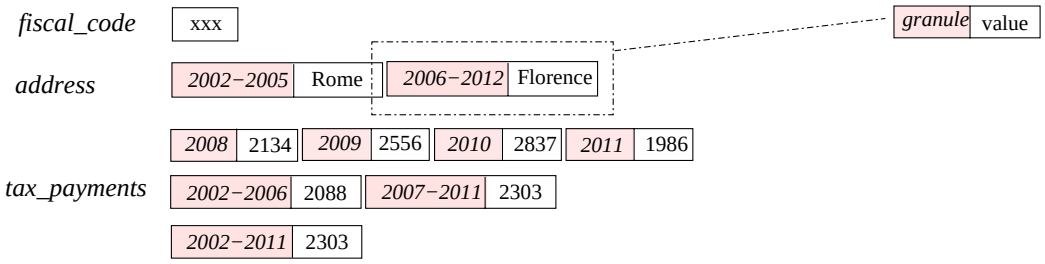


FIGURE 1. Example of object state

2.3 T_ODMGe object consistency

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 provided for each attribute. In T_ODMG this notion has been revisited in order to take into account 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 dynamic attributes. Intuitively, the constraints that these attributes should meet are the following:

(i) the value of a dynamic attribute must be a tuple of temporal values; each value in the tuple, in turn, must be a legal value for the corresponding temporal type in the attribute domain in the class definition;

(ii) for each specified value, the temporal interval associated with the corresponding granule must intersect the object lifespan;

(iii) temporal values in the tuple, for temporal value restrictions corresponding to the same portion of the time domain, must be related by the corresponding coercion function, if these values are defined.

The following definition formally states all the constraints an object should meet in order to be a valid instance of a T_ODMGe class. Note that conditions (1), (2.a), and (2.b) are inherited from T_ODMG while condition (2.c) has been added for dynamic attributes.

DEFINITION 2.5 (T_ODMGe consistent instance)
Let $o = (id, N, (a_1 : v_1, \ldots, a_p : v_p), c, [i_o, j_o]^{G_I})$ be an object. Let c be a class and $attr$ its attribute specification. Object o is a consistent instance of c if the following conditions hold:

1. $\forall\, i, 1 \leq i \leq p, \exists\, (\tau, a, coerc) \in attr$ such that $a = a_i$;
2. $\forall\, (\tau, a, coerc) \in attr, \exists k, 1 \leq k \leq p$, such that $a = a_k$ and the following conditions hold:
   (a) if a_k is a static attribute (i.e. $\tau \in \mathcal{LT} \cup \mathcal{OT}$): $v_k \in [\![\, \tau \,]\!]_{now}$;
   (b) if a_k is a temporal attribute (i.e. $\tau \in \mathcal{TT}$ and τ = *temporal*$_G(\tau')$) all the following conditions hold: $v_k \in [\![\, \tau \,]\!]$, i.e. the attribute value belongs to the set of legal values of the corresponding type; and $\forall i \in \mathcal{IS}$ such that $v_k(i)$ is defined, $G(i) \cap [i_o, j_o]^{G_I} \neq \emptyset$;[4] i.e. for each value defined, the corresponding temporal granule intersects the object lifespan;

[4] This is a short notation for $G(i) \cap (\bigcup_{i_o \leq k \leq j_o} \{G_I(k) \mid k \in \mathcal{IS}\}) \neq \emptyset$.

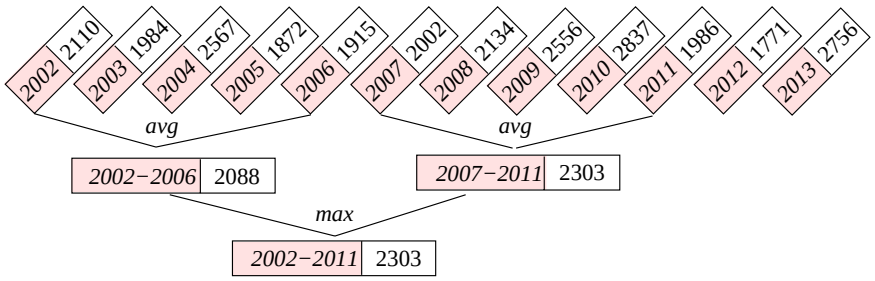


FIGURE 2. Example of application of coercion functions at different levels of detail

(c) if a_k is a dynamic attribute (i.e. $\tau = (\tau_1, \ldots, \tau_n)$, such that $\forall j,\ 1 \leq j \leq n$, $\tau_j = temporal_{G_j}(\tau')$, $\tau' \in \mathcal{LT} \cup \mathcal{OT}$, $G_1 \prec \ldots \prec G_n$, and $(C_1, \ldots, C_{n-1})$ is the corresponding tuple of coercion functions),[5] the following conditions hold:

(i) $v_k = (v_{k_1}, v_{k_2}, \ldots, v_{k_n})$, where $\forall j, 1 \leq j \leq n$, $v_{k_j} \in [\![\tau_j]\!]$;

(ii) $\forall j, 1 \leq j \leq n$, $\forall i \in \mathcal{IS}$ such that $v_{k_j}(i)$ is defined, $G_j(i) \cap [i_o, j_o]^{G_I} \neq \emptyset$;

(iii) $\forall j, 2 \leq j \leq n, \forall i \in \mathcal{IS}$ such that $v_{k_j}(i)$ is defined, let $\Upsilon_i^{G_{j-1}} = \{G_{j-1}(h_1), \ldots, G_{j-1}(h_m)\}$ $= \{G_{j-1}(h_k) \mid h_k \in \mathcal{IS}, G_{j-1}(h_k) \subseteq G_j(i)\}$ be the temporal element that contains all the granules of granularity G_{j-1} included in $G_j(i)$. If all values $v_{k_{j-1}}(h_1)$, $\ldots$,$v_{k_{j-1}}(h_m)$ are defined, then $v_{k_j}(i) = C_{j-1}(v_{k_{j-1}|\Upsilon_i^{G_{j-1}}})(i)$.

EXAMPLE 2.6
Consider the class `taxpayer` of Example 2.2 and the object i_1 of Example 2.4. We focus on the `tax_payments` attribute, that we report in Figure 2, since it is the only dynamic attribute defined for i_1. Note that, in Figure 2 we depict also the oldest values, corresponding to payments of years from 2002 to 2007. The value of the `tax_payments` attribute is a triple of temporal values at granularities $years$, $5years$, and $decades$, respectively. Condition (2.c-i) is met, because each temporal value is a legal value for the corresponding temporal type. Condition (2.c-ii) is met as well, because each defined value refers to a temporal interval (identified by the corresponding granule) that intersects the object lifespan. Condition (2.c-iii) also holds, because the temporal value at granularity $5years$ has been obtained by applying coercion function avg to the temporal value at granularity $years$, whereas the temporal value at granularity $decades$ has been obtained by applying coercion function max to the temporal value at granularity $5years$. The application of the coercion functions and the corresponding results are shown in Figure 2. Therefore, object i_i is consistent with the definition of the `taxpayer` class.

The above consistency notion requires that, if data for a dynamic attribute are available/defined, for every instant of the object lifespan they should be coherent. In other words, for each pair of contiguous temporal values at different granularities in the attribute value, the value at the coarser granularity must be obtained by applying the coercion function to the corresponding values at the finer granularity. However, such a constraint does not apply when data at the finer granularity are not available, as in the case they have been deleted. This behaviour is pointed out by the two states of the `tax_payements` attribute reported in Figure 1 and Figure 2. We will motivate this behaviour

[5] We recall that, for each $j,\ 1 \leq j \leq n-1$, $C_j : [\![\tau_j]\!] \rightarrow [\![\tau_{j+1}]\!]$.

```
⟨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 every 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 3. BNF of attribute declaration

in Section 5.4, where we discuss issues concerning the management of value deletion. Moreover, in Section 5.3, we will show that, when information at a certain level of detail is missing, consistency is assured by the semantics of the update operations for dynamic attributes.

3 Specification of expiration for dynamic attributes

Conditions for attribute expiration often depend on attribute semantics and are *a priori* known, therefore they can be conveniently specified together with the schema. For this reason the $T_$ODMG class definition language has been extended in order to support the declarative specification of dynamic attributes. In $T_$ODMGe, the temporal type of a dynamic attribute is specified as for a usual temporal attribute. Additionally, for such an attribute one can specify: the various granularities at which the temporal values should be represented, that is, for each detail level, the coercion function to convert the attribute value to the immediately coarser level; the possible deletion operations; and, finally, the frequencies at which expiration operations must be executed. For each value deletion specified for the attribute, we require also the amount of data to be deleted. Since granularity evolution and value deletion are orthogonal operations, they are specified separately by means of `evolve` and `delete` clauses that are discussed in detail in the remainder of this section.

Figure 3 shows the BNF grammar for the declaration of dynamic attributes. 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 static type name. In the remainder of the section, we first present the syntax to specify evolutions. We then describe the syntax for the deletion specification, and finally conditions ensuring that the specification of a dynamic attribute is consistent are discussed.

3.1 Specification of granularity evolutions

An attribute granularity evolution is specified by means of the `evolve` clause. Such a clause states the target granularity of an evolution operation, the frequency at which the evolution should be executed, and the coercion function to apply. More than one `evolve` clause can be specified for the same attribute, each with a different target granularity. The number of `evolve` clauses declared for an attribute corresponds to the number of different representations, one for each target granularity, at which the user wishes to represent the attribute. Coercion functions that we can use in an `evolve` clause are those presented in Section 2.1. If more than one `evolve` clause is specified for the same attribute, the temporal value specified at a certain granularity is used to compute the value at the immediately coarser granularity.

EXAMPLE 3.1
Consider the class specification of Example 2.2. Attribute `tax_payments` represents payments of taxes of an accountant customer and is defined at granularity $years$. As stated in the description of the running example, after five years, we would know the average tax payment over the previous years, and the same for the following years. Then, after ten years we would also know the maximum average amount, referred to the previous ten years. These application requirements can be directly supported by the following definition for the `tax_payments` attribute:

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

In the language there are no constraints about the order in which the `evolve` clauses are specified. Their order can be determined relying on the $\prec$ relationship that exists among granularities.

3.2 Specification of deletions of values

The deletion of values of a dynamic attribute is specified by means of the `delete` clause. Each `delete` clause specifies the temporal value and the frequency according to which the deletion should be performed. The frequency specification is analogous to the one for evolution. The amount of data to be deleted is specified as a temporal period.[6] The values removed from the database are the oldest that appear in the temporal value.

EXAMPLE 3.2
Consider Example 3.1 and the application requirement that after five years we wish to remove the oldest values corresponding to one year of tax payments from the temporal value at granularity $years$. Then, the following declaration should be added to the attribute definition of Example 3.1:

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

Deletions can be specified for both the temporal value at the base granularity (as in the previous example) and the other temporal values that represent the attribute granularity 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 each dynamic attribute.

3.3 Consistency constraints for dynamic attribute specifications

The T_ODMGe specification language in Figure 3 allows one to specify evolution and deletion frequencies that do not make sense. Consider for instance the following example.

EXAMPLE 3.3
Consider the following `evolve` clause:

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

specified for an attribute defined at granularity $days$. Such a specification is correct according to the T_ODMGe BNF grammar, but the evolution this clause expresses is not meaningful, because before obtaining a monthly value we need data that refer to a period of month, and not to a single day.

[6] The semantics of temporal periods used in T_ODMGe is given in Section 4.2.

To avoid such types of definition, that can result in an inconsistent database schema, some additional conditions are imposed on T_ODMGe expiration specifications to ensure their consistency. Those constraints reflect the semantics of evolution and deletion operations and cannot be expressed in the BNF grammar.

However, we are not always able to relate a frequency specification with a different granularity, or to compare different temporal periods, thus for certain specifications we cannot decide whether they are consistent or not. This problem is due to the fact that the only relationship the model admits on granularities is the finer-than relationship. Relying only on this relationship, given two granularities G and H, with $G \prec H$, we cannot establish how many G granules we need to obtain a H granule. Note that such a situation arises for very common granularities, such as $months$ and $days$, since we cannot state how many days are needed to obtain a month. However, also for granularities that do not pose any problem, the finer-than relationship does not tell us, for example, that 24 $hours$ are needed to obtain 1 $days$,[7] as showed by the following example.

EXAMPLE 3.4
Consider the following `delete` clause:

`delete` 5 $days$ `from` $days$ `after` 100 $hours$;

Such a specification is ambiguous, because according to the finer-than relationship, we cannot decide whether 100 $hours$ are enough to obtain 5 $days$. In the same way, we cannot automatically check the consistency of the following `evolve` clause:

`evolve to` $days$ `every` 150 $hours$;

although we know that such an evolution specification is semantically correct, since each granule of granularity $days$ can be regularly shared in granules of granularity $hours$.

We refer to temporal periods used in `evolve` and `delete` clauses that are not comparable in T_ODMGe model as *incomparable temporal periods*. We can devise two different cases of periods that we are not able to compare due to the assumptions made in the T_ODMGe model. The first case arises when the granularities used to express the periods are not related in any way by finer-than relationship. The second situation arises when, although the granularities involved are comparable, we compare a bigger number of granules at the finer granularity with a small number of granules at coarser granularity, like in the previous Example, in which we required to compare 150 $hours$ and 5 $days$. The following definition formalizes this notion.

DEFINITION 3.5 (Incomparable temporal periods)
Two temporal periods n_G G and n_H H with, $n_G, n_H \in \mathbb{N}$, $n_G > 0$, $n_H > 0$, $G, H \in \mathcal{G}$ granularities, are *incomparable* if one of the following conditions holds:

- $G \not\preceq H$ and $H \not\preceq G$.
- $H \prec G$ and $n_H > n_G$;
- $G \prec H$ and $n_G > n_H$.

In the remainder of the section, we detail the additional constraints imposed to obtain a consistent T_ODMGe database schema. Section 3.3.1 describes constraints for evolution specifications, whereas Section 3.3.2 describes those for deletions. In both sections we refer to the generic attribute declaration for a dynamic attribute reported in Figure 4, where: $m_1, \ldots, m_{n-1}, p_1, \ldots, p_n$, $q_1, \ldots, q_n$ are $3n-1$ natural numbers greater than 0; $G_1, \ldots, G_n$ are n distinct granularities; $H_1, \ldots, H_{n-1}$, $K_1, \ldots, K_n$, $J_1, \ldots, J_n$ are $3n-1$ granularities; $\tau \in \mathcal{LT} \cup \mathcal{OT}$ is a literal or object type; and $cf_1, \ldots, cf_{n-1}$ are $n-1$ coercion functions.

[7] This issue could be *partially* addressed introducing other well-known relationships among granularities (e.g. the *partition* relationship) [3]

```
attribute temporal G1(τ) a{
    evolve to G2 every m1 H1 using cf1;
    delete p1 J1 from G1 after q1 K1;
    ...
    evolve to Gn every mn−1 Hn−1 using cfn−1;
    delete pn−1 Jn−1 from Gn−1 after qn−1 Kn−1;
    delete pn Jn from Gn after qn Kn;
};
```

FIGURE 4. Generic attribute declaration with evolutions and deletions

3.3.1 Consistency constraints for granularity evolution specification

In a granularity evolution specification there are some relationships among granularities and frequencies that should be satisfied in order to have a consistent evolution specification. In the remainder of the section we deal with them and we present their formalization.

Relationship between the base and the target granularity. The most important characteristic of evolution is to reduce the level of detail for temporal information as time passes, that is to convert oldest data to a coarser granularity. Thus, a first condition is that *the evolution must be performed from a finer granularity to a coarser one.*

EXAMPLE 3.6
Consider the dynamic attribute specification of Example 3.1. The first `evolve` clause specifies an evolution from granularity $years$ to granularity $5years$, and the second one specifies an evolution from granularity $5years$ to granularity $decades$. This condition is satisfied because $years \prec 5years \prec decades$. Consider now the following specification for a dynamic attribute:

```
attribute temporal_days(float) temperature {
   evolve to weeks every 5 weeks using main;
   evolve to months every 12 months using max;
};
```

This specification does not meet the constraint because $weeks \not\prec months$.

Relationship between the frequency and the target granularity. A second constraint is that the evolution frequency must be as coarse as to allow the collection of enough data to perform the evolution. For instance, if we want to evolve data to granularity $years$, we must wait for a year before performing the first evolution, for two years before performing the second evolution, and so on, thus the frequency cannot be finer-than 1 $years$. We express this constraint stating that *the evolution frequency should allow the collection of enough data to perform the evolution.* Note that this condition allows one to obtain dynamic attributes that meet the consistency constraints imposed by Definition 2.5.

EXAMPLE 3.7
Consider the attribute specification for a best seller item in a commercial activity:

```
attribute temporal_days(string) best_sellers {
  evolve to bweeks every 1 weeks using main;
};
```

This `evolve` clause specifies an evolution from granularity $days$ to granularity $bweeks$ (that is the business weeks granularity corresponding to the days from Monday to Friday), that is performed every week. This specification is correct because after one week, we are sure to have collected enough data to perform such an evolution.

(1e)	$\forall i,\ 1 \leq i \leq n-1,\ G_i \prec G_{i+1}$
(2e)	$\forall i,\ 1 \leq i \leq n-1,\ G_{i+1} \preceq H_i$
(3e)	$\forall i,\ 1 \leq i \leq n-2,\ H_i \prec H_{i+1}$ `and` $m_i \leq m_{i+1}$

FIGURE 5. Constraints for `evolve` clauses

Relationship between frequencies of different evolutions. In order to have a meaningful chain of evolution operations, the evolutions must be executed with coarser and coarser frequencies, so that step by step coarser values are obtained. This condition can be expressed by saying that *the evolution frequency of each evolution operation must be coarser than the one of the preceding evolution*, taking one granule at the base granularity as the frequency of evolution of the base granularity itself.

EXAMPLE 3.8
Consider the dynamic attribute specification of Example 3.1. The first `evolve` clause specifies an evolution frequency that causes the insertion of a new value at granularity $5years$ every 5 $years$, whereas the second one specifies an evolution frequency that inserts a new value at granularity $decades$ every 10 $years$. The constraint is met because the second `evolve` clause frequency specifies a period greater than the one specified by the first clause.

Consistency conditions for evolution specifications. Given an attribute definition according to the structure in Figure 4, the constraints reported in Figure 5 formalize the previous conditions, and summarize the conditions that a granularity evolution should meet in order to have a meaningful dynamic attribute definition. Constraint (2e) assures that we have enough data to perform the evolution. Note also that the first and the third conditions express the idea that data become less important as they age. Indeed, first we summarize oldest data at a certain level of detail. Then, by taking into account that the number of granules denoted by m_i is greater than zero, after an amount of time that corresponds to the difference between the two evolution frequencies (3e), during which less detailed data become again less interesting, we further summarize these data at a coarser level of representation (1e). Evolution clauses that specify incomparable temporal periods are simply rejected by the T_ODMGe specification interpreter.

3.3.2 Consistency constraints for value deletion specification

Also for deletion specifications a set of constraints can be devised ensuring that a T_ODMGe database schema is consistent. In the remainder of the section they are discussed and formalized.

One deletion per each level of detail. First of all, we want to prevent ambiguous or redundant specification of values deletions. Thus, as in the case of granularity evolution specifications discussed in the previous section, the definition of a dynamic attribute may only include *at most one deletion specification for each representation level*.

Relationship between frequency and granularity. A fundamental requirement is that deletion operations be consistent with the semantics of granularities. Granularities express integrity constraints on temporal values, because they establish their finest update frequency. Then, we should assure that the deletion operation, which is a particular form of update, also meet this constraint. Thus, we avoid deletion specifications that require one to remove, from a temporal value, values corresponding to portion of granules, by imposing the constraint: *the amount of deleted data* [8] *is specified by a*

[8] We recall that in a `delete` clause, the amount of data that should be deleted is specified through a generic temporal period that refers the granules associated to values in a temporal value.

granularity coarser than or equivalent to the granularity at which the temporal value is expressed.

EXAMPLE 3.9
The following `delete` clause violates the constraint just given:

`delete` 20 $seconds$ `from` $days$ `after` 1 $days$;

because it is forbidden to delete an amount of 20 seconds from a temporal value specified at granularity $days$, since this will mean considering portions of $days$ granules.

Relationship between deletion and evolution frequency. An obvious aspect to consider for deletions is that we cannot delete a value not stored in the database, i.e. *the frequency at which data are deleted should be coarser than the frequency at which data are collected*. In other words, for each level of detail for which a deletion operation has been specified (including the temporal value at the base granularity), the evolution operation (the insertion operation, for the temporal value at the base granularity) must be performed more frequently than the corresponding deletion operation.

EXAMPLE 3.10
Consider the following attribute specification:

```
attribute temporal_days(string) best_sellers {
  evolve to months every 5 months using main;
  delete 1 months from months after 2 months;
};
```

Such a specification does not meet the constraint because it requires deleting more values from granularity $months$ than those stored.

Relationship between deletion frequency and amount of deleted data. We require that *the amount of deleted data should be smaller than or equal to the frequency at which the same data are deleted*.

EXAMPLE 3.11
Consider the following attribute specification:

```
attribute temporal_days(string) best_sellers {
  delete 5 days from days after 1 days;
};
```

After 1 $days$, the value for attribute `best_sellers` will be empty, and it will remain empty forever. This is because each value inserted for attribute `best_sellers` will immediately be deleted.

Consistency conditions for deletion specifications. Given an attribute definition with the structure given in Figure 4, Figure 6 summarizes and formalizes the conditions that must be satisfied to have consistent deletion clauses. The first condition (1d) formalizes the requirement that the amount of data to be deleted must be specified by using a granularity that is equal or coarser-than the granularity specified for the temporal value, thus preventing the deletion of values that refer granule portions. The relationship that must be verified between deletion and evolution frequencies is formalized by condition (2d), while condition (3d) expresses the requirement that the amount of data removed must not be greater than the frequency specified for the deletion operation.

4 Expiration execution: preliminaries

In this section we discuss some preliminary concepts that will be exploited in Section 5 to present the execution model of the expiration operations: granularity evolution and value deletion. Specifically,

(1d)	$\forall i,\ 1 \leq i \leq n-1,\ G_i \preceq J_i$
(2d)	$\forall i,\ 0 \leq i \leq n-1,\ m_i \leq q_{i+1},\ H_i \preceq K_{i+1},$ with $m_0 = 1$ and $H_0 = G_1$
(3d)	$\forall i,\ 1 \leq i \leq n-1,\ p_i \leq q_i,\ J_i \preceq K_i$

FIGURE 6. Constraints for `delete` clauses

Section 4.1 introduces the temporal dimension T_ODMGe inherits from T_ODMG model and the issues concerning the management of the expiration operations in such dimension. Then, Section 4.2 deals with the different interpretations of the notion of *temporal periods* that can be taken into account in a multigranular temporal model and the one T_ODMGe adopts. Finally, Section 4.3 compares two different approaches for evaluating the conditions to trigger the evolution and deletion processes.

4.1 Valid time as temporal dimension

T_ODMGe inherits from T_ODMG the assumption of valid time as the reference temporal dimension. This means that T_ODMGe temporal values are stored into the database considering the time at which they are valid/true in reality, instead of the time in which they are stored in the database. [9] For instance, if we say that Mr. Green lived in Florence during year 2000, the time related to Mr. Green's address is 2000 even if such information has been stored in the database, for example, during year 2002.

The introduction of a mechanism for granularity evolution and deletion of values by means of frequencies in a data model based on valid time dimension raises several issues. First of all, we must consider the *management of nonmonotonic updates*. The possibility to update values already inserted in the database complicates the management of the expiration operations. How our mechanism handles nonmonotonic updates will be discussed in Section 5. Moreover, *when can we check whether an expiration condition is verified?* In a model based on transaction time, such a moment can be specified by the variable `now`, that refers to the current time instant, or as the moment in which an attribute is accessed. However, these moments are 'intrinsically transactional' and cannot be exploited in a data model based on valid time. In Section 4.3 we will discuss two possible approaches for determining when to check expiration conditions in a valid time model.

Note that expiration operations can be conveniently expressed in a bi-temporal data model, thus exploiting both the valid and transaction time as temporal dimensions. However, in this paper we address the issues raised by the introduction of the expiration mechanism in a data model based on valid time and we left the investigation of expiration in a bi-temporal model as future work.

4.2 Semantics of temporal periods

Temporal periods, representing frequencies specified both in `evolve` and `delete` clauses and the amount of data to delete specified in `delete` clauses, can be interpreted in two different ways.

The interpretation adopted in [20, 21] corresponds to the one used in natural language. The most intuitive interpretation of a phrase like '2 $months$' is 'a temporal interval of two months that starts at the instant when the phrase is expressed'.

The T_ODMGe model adopts a different interpretation, that is more consistent with the standard notion of granularity we refer to. Since a granularity makes a partition of the time domain into

[9] This is called transaction time.

indivisible portions (i.e. granules) we mean only whole granules. Then, the phrase '2 $months$' is interpreted as 'two calendar months from the beginning of the month during which the phrase is expressed'.

Then, the phrase '2 $months$' always identifies a temporal period, but we can obtain different results on expiration execution if we follow one interpretation rather than another, as the following example shows.

EXAMPLE 4.1
Suppose the expression '2 $months$' is used to specify an evolution frequency of a dynamic attribute, and the first value for this attribute was stored on 15th January. Following the first interpretation, '2 $months$' later means 15th March. By contrast, following the second interpretation, '2 $months$' later means 1st March.

4.3 Verification of the expiration condition: lazy vs quantum approaches

The expiration process is activated when an *expiration condition* is verified. The issue we address in this section is to find the most appropriate moment for checking such a condition. Note that, since we consider valid time of data, the only operations, that should be checked for verifying the event that triggers the expiration, are related to the operations of insertion or update of a temporal value in a dynamic attribute.

The verification of the event can be performed following a 'transactional approach', i.e. by fixing a quantum of time every time the event is checked, or an 'operational approach', i.e. by checking the verification of the event on each insertion/update performed on a dynamic attribute. The first approach is referred to as *quantum approach*, whereas the second one as *lazy approach*.

According to the quantum approach, at established quanta of time, the database engine checks, for each dynamic attribute in the database, if some expiration condition is verified. Then, it executes the expiration process on all the attributes for which the condition is verified. Since the expiration operation may involve many objects, its execution can lead the database to

- become inaccessible for a long period of time and, consequently, to interrupt database application that are currently accessing/updating the objects stored in the database;
- rollback the transactions that are accessing the objects involved in expiration operations.

Moreover, if the quantum is too long, dynamic attributes may be inconsistent in some instants of time, because their value could be evolved or deleted, but the quantum of time is not elapsed.

By contrast, according to the lazy approach, the database engine checks the expiration condition each time a new/updated value is inserted/updated in the database for a dynamic attribute. Whenever the condition is verified, it executes the expiration operation only on this attribute. The performance of the expiration operation improves, because only attributes involved in insertions and updates are considered. However, the performance of insertion and update operations are affected by the execution of the expiration operation.

T_ODMGe executes expiration operations according to the lazy approach, because the consistency of dynamic attributes at different granularities is always guaranteed. Moreover, this approach is particularly convenient for dynamic attributes that are not updated regularly, for which the database stores only partial information.[10] Finally, rollback of transactions is limited because the evolution and deletion operations are performed for one dynamic attribute at a time.

[10] T_ODMGe supports this situation. Indeed, a granularity is a constraint that expresses only the maximum frequency of update of a temporal value. Then, less frequent updates are admitted.

5 Expiration execution

This section describes how expiration operations specified for a dynamic attribute are performed. Section 5.1 describes the general expiration process. Section 5.2 presents how nonmonotonic updates are handled. Section 5.3 formalizes the semantics of update operation for dynamic attributes, focusing on the expiration execution. Finally, Section 5.4 briefly discusses the impact of value deletion on object consistency.

5.1 Expiration process

Values of a dynamic attribute are always inserted and updated at the finest among the granularities at which the attribute value is maintained, that is the attribute base granularity.

When a value for a dynamic attribute is inserted in the database, in order to establish if some expiration operation should be performed, the system considers the *attribute starting time*. For each level and for each expiration operation, the starting time is the valid time of the older value in a temporal value that has not been considered by preceding executions of the operation. Specifically, for an evolution operation, the starting time is the valid time of the older value that has not partecipated yet to the granularity evolution. By contrast, the starting time for a deletion operation is the valid time of the oldest value that has not been involved by preceding deletions yet.

Note that values are not necessarily inserted in the database in a regular way, that is as a stream, and, moreover, that values for some valid time can never be inserted and that they can be deleted from the database. Then, the starting time can refer to the valid time for a value that is not present in the database, as pointed out by the following example.

EXAMPLE 5.1
Considering the running example, if we insert into the database values corresponding to tax declarations of year 2002 and then of year 2007, after having performed the first deletion from temporal value at granularity $years$, the starting time for deletion operation is set to granule '2003', although no value has been inserted into the database referring to this granule.

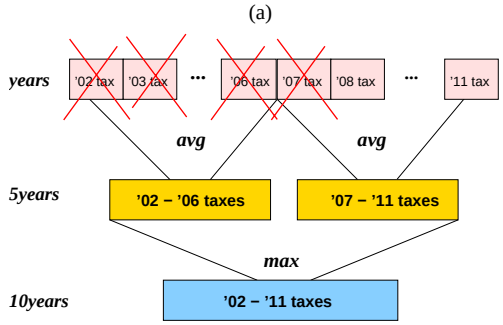
Then, when a value of a dynamic attribute is inserted and the expiration condition holds, the expiration process is performed. Evolution operations are performed before deletion. Both operations are performed considering the order imposed on temporal values considering their granularities and respect to finer-than relationship, starting from the temporal value at the base granularity and then considering the immediately coarser one, and so on. For each temporal value v of type $temporal_G(\tau)$ and for each operation specified for v,[11] the system considers the temporal interval bounded by the starting time of v and the valid time of the inserted value. If this temporal interval is greater than or equal to the frequency specified for the operation, the operation is performed.

If the operation is a granularity evolution, the system then considers the temporal element, let it be Υ^{G_t}, at the evolution target granularity G_t, fully contained into the temporal interval bounded by the starting time of v and the valid time of the inserted value. If this temporal element is not empty, that is it contains at least one granule at granularity G_t,[12] the system applies the coercion function specified in the corresponding `evolve` clause to the temporal value $v_{|_{\Upsilon^{G_t}}}$, that is the temporal restriction of v to the temporal element Υ^{G_t}. The value obtained is inserted in the temporal value at the evolution target granularity G_t, that is the temporal value at the granularity immediately coarser than that of 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.

[11] We recall that each operation refers to a specific temporal value v.
[12] It can be considered as a period with respect to the semantics expressed in Section 4.2.

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

(a)



(b)

FIGURE 7. (a) Class declaration and (b) evolution and deletion process

If the operation is a deletion, by contrast, the system deletes the oldest data, if defined, from the temporal value v, as specified in the corresponding `delete` clause. The expiration process just explained is described in the following Example.

EXAMPLE 5.2
Figure 7(a) shows the class specification for the tax payment running example, whereas Figure 7(b) is a sketch of what happens to the `tax_payments` attribute value over a period of ten years, considering the object state depicted in Figure 1. In Figure 7(b) we highlight the deletion of oldest values. Actually, instead of the full temporal values, we represented only granules for defined values.

When the first value at granularity $years$ for attribute `tax_payments` is inserted in the database, i.e. the value corresponding to 2002 declaration, we set both evolution and deletion starting times of $years$ level to granule '2002'. This value is not updated until the tax declaration corresponding to year 2006 is inserted. When the value corresponding to year 2006 is inserted, the length of the temporal interval bounded by the starting time and the valid time of this value is equal to the temporal period specified by $5years$ frequency of the first `evolve` clause. Therefore, the first granularity evolution from granularity $years$ to granularity $5years$ is performed. The coercion function avg is applied to the temporal value restriction $v_{|_{\Upsilon^{5years}}}$, with $\Upsilon^{5years} = \{$'2002-2006'$\}$, that is the temporal value $\{\langle$'2002',2110$\rangle, \langle$'2003',1984$\rangle, \langle$'2004', 2567$\rangle, \langle$'2005', 1872$\rangle, \langle$'2006',1915$\rangle\}_{years}$.

The coercion function returns the temporal value $\{\langle$'2002-2006',2088$\rangle\}_{5years}$, that is inserted in the representation level at granularity $5years$ of the attribute. Moreover, the evolution starting time at granularity $years$ is updated to granule '2007', and that of granularity $5years$ is set to granule '2002-2006'. Finally, the first deletion operation specified for value at granularity $years$ is executed. The oldest value, corresponding to year 2002, is deleted and the deletion starting time is updated to granule '2003'.

For deletions, the system works in the same way for the tax declarations of the remaining years. When the tax declaration corresponding to year 2007 is inserted, we delete the value with valid time '2003' from temporal value at granularity $years$, updating the deletion starting time to '2004'. By contrast, the next evolution is executed when we insert the value corresponding to 2011 declaration, performing the second evolution to granularity $5years$. The temporal element $\Upsilon^{5years} = \{$'2006–2011'$\}$ is considered and the coercion function avg is applied to the temporal value restriction $v_{|_{\Upsilon^{5years}}} = \{\langle$'2007', 2002$\rangle, \langle$'2008', 2134$\rangle, \langle$'2009', 2556$\rangle, \langle$'2010', 2837$\rangle, \langle$'2011', 1986$\rangle\}_{\,years}$. Therefore, the temporal value at granularity $5years$ is updated to the value $\{\langle$'2002–2006', 2088$\rangle$, $\langle$'2006–2011', 2303$\rangle\}_{5years}$, and '2012' became the new evolution starting time of the temporal value at granularity $years$.

As a consequence of this evolution, the evolution specified by the second `evolve` clause can also be performed, because the length of the temporal interval bounded by the starting time of temporal value at granularity $5years$ and the valid time of the value just inserted is equal to the frequency of 10 $years$ specified for this evolution. The evolution from granularity $5years$ to granularity $decades$ is performed by applying the coercion function max to the temporal value restriction $v'_{|_{\Upsilon decades}}$ with $\Upsilon^{decades} = \{$'2002–2011'$\}$, that is the temporal value $\{\langle$'2002–2006', 2088$\rangle$,$\langle$'2006–2011', 2303$\rangle\}_{5years}$, as done for the previous granularity. The application of the coercion function max returns the temporal value $\{\langle$'2002–2011', 2303$\rangle\}_{decades}$, and the evolution starting time of the temporal value at granularity $5years$ is updated to '2012–2016'. Finally, as highlighted in Figure 7(b), we execute the 5th deletion from $years$ granularity.

5.2 Nonmonotonic updates of dynamic attributes

Since T_ODMGe is a valid time model, updates can be performed in a nonmonotonic way with respect to time, that is the valid time of an updated value can precede one or more valid times of values already stored for the attribute. Nonmonotonic updates are meaningful only for temporal and dynamic attributes. The former attributes do not pose any problem with respect to nonmonotonic updates. The value is inserted into the temporal value if none of the values present has the same valid time of the updated value; whereas, if a value in the temporal value exists with the same valid time, it is updated. By contrast, a nonmonotonic update of a dynamic attribute may require the update of coarser temporal values in order to maintain all the attribute representation levels consistent according to Definition 2.5.

In the preliminary version of the model, presented in [8], nonmonotonic updates were allowed only for recent values of a dynamic attribute, i.e. values whose valid times are more recent then the evolution starting time of the base granularity of the attribute. This constraint forces updates to dynamic attributes to be monotonically executed for all values preceding the starting time of the base granularity. We called this constraint *update monotony restriction*. By contrast, in this paper we propose an extension of the model, supporting updates of values whose valid times are older than the evolution starting time of the base granularity, though still with some restrictions that we discuss later on this section.

Propagation of the update effects to coarser levels of a dynamic attribute could require one to recompute the evolutions that have already been performed — potentially an expensive operation. Luckily, depending on which coercion function has been used to obtain the coarser representation level, in most cases we are able to optimize this re-computation, minimizing data needed to perform it, and then also the accesses to the database. Sometimes, to perform this *smart* re-computation, we need to maintain some additional information.

TABLE 1. Propagation of nonmonotonic update effects to evolution levels

c.f.	Insertion	Update
sum	$v^c_{new} = v^c_{old} + v^f_{new}$	$v^c_{new} = v^c_{old} - v^f_{old} + v^f_{new}$
avg	$v^c_{new} = \frac{v^c_{old} * count + v^f_{new}}{count+1}$, $count = count + 1$	$v^c_{new} = \frac{v^c_{old} * count - v^f_{old} + v^f_{new}}{count}$
min	$v^c_{new} = \begin{cases} v^f_{new} & if\ v^f_{new} < v^c_{old} \\ v^c_{old} & otherwise \end{cases}$	$v^c_{new} = \begin{cases} v^f_{new} & if\ v^f_{new} \leq v^c_{old} \\ v^c_{old} & if\ v^f_{new} > v^c_{old}\ and\ v^c_{old} \neq v^f_{old} \\ \perp_p & if\ v^f_{new} > v^c_{old}\ and\ v^c_{old} = v^f_{old} \end{cases}$
max	$v^c_{new} = \begin{cases} v^f_{new} & if\ v^f_{new} > v^c_{old} \\ v^c_{old} & otherwise \end{cases}$	$v^c_{new} = \begin{cases} v^f_{new} & if\ v^f_{new} \geq v^c_{old} \\ v^c_{old} & if\ v^f_{new} < v^c_{old}\ and\ v^c_{old} \neq v^f_{old} \\ \perp_p & if\ v^f_{new} < v^c_{old}\ and\ v^c_{old} = v^f_{old} \end{cases}$
all	$v^c_{new} = \begin{cases} v^f_{new} & if\ v^f_{new} \neq v^c_{old} \\ v^c_{old} & otherwise \end{cases}$	$v^c_{new} = \begin{cases} v^c_{old} & v^c_{old} \neq \perp\ and\ v^f_{new} = v^c_{old} \\ \perp_p & v^c_{old} = \perp\ or\ v^f_{new} \neq v^c_{old} \end{cases}$
$first$	$v^c_{new} = \begin{cases} v^f_{new} & if\ i < j \\ v^c_{old} & otherwise \end{cases}$	$v^c_{new} = \begin{cases} v^f_{new} & if\ i = j \\ v^c_{old} & otherwise \end{cases}$
$last$	$v^c_{new} = \begin{cases} v^f_{new} & if\ i > j \\ v^c_{old} & otherwise \end{cases}$	$v^c_{new} = \begin{cases} v^f_{new} & if\ i = j \\ v^c_{old} & otherwise \end{cases}$
$main$	$v^c_{new} = \begin{cases} v^c_{old} & if\ v^f_{new} = v^c_{old} \\ \perp_p & otherwise \end{cases}$	$v^c_{new} = \begin{cases} v^c_{old} & if\ v^f_{new} = v^c_{old} \\ \perp_p & otherwise \end{cases}$
$proj$	$\perp_p$	$v^c_{new} = \begin{cases} v^f_{new} & if\ i = j \\ \perp_p & otherwise \end{cases}$

Since for each update only one value[13] for each representation level needs to be updated,[14] we can consider two contiguous representation levels at a time, specifying the computation to apply to the value at the finer granularity. The operations that can be performed are summarized in Table 1, and are applied as follows. First, a suitable re-computation is applied considering the temporal value at base granularity and the immediately coarser level. Then, if more than one `evolve` clause is declared for the attribute, the update effects are propagated by taking the latter value as the finer value and its immediately coarser value as the coarser one, and by applying the right operation to these values. These steps are executed for all the evolution levels that must be updated.

In Table 1, referring to a dynamic attribute for which more than one evolution has been specified, $\langle v^f_{new}, G_f(i)\rangle$ is the updated value at the finer granularity G_f, and $\langle v^f_{old}, G_f(i)\rangle$ is the same value before the update (then i is the granule index at granularity G_f that represents the valid time of the updated value); $\langle v^c_{old}, G_c(t)\rangle$ is the coarser value that must be affected by the update, and $\langle v^c_{new}, G_c(t)\rangle$ is the same value after propagation of the update. Note that the temporal interval identified by granule $G_f(i)$ is contained in the temporal interval identified by the $G_c(t)$ as stated by the finer-than relationship. $count$ is additional information that should be maintained for coarser values obtained by applying the avg coercion function and represents the number of values from which the average has been obtained. Knowing the value of $count$, one does not need to recompute the full coarser temporal value, thus the performance of the avg coercion function is greatly improved.

Moreover, if the value has been obtained using the $first$ or $last$ coercion functions, both insertion and update performances can be improved by knowing the granule index j at granularity G_f from which value v^f_{old} has been obtained. Knowing the value of the G_f granule index improves the performance of updates of coarser values that result from the application of the $proj$ coercion function. Finally, $\perp_p$ denotes that we are not able to update the coarser value starting from these few values, but we must reapply the specific coercion function to the finer temporal value restriction that corresponds to the coarser granule.

[13] Note that this is a single value, that is the temporal value restriction obtained referring to one granule of the granularity of the temporal value.

[14] This follows from the definition of finer-than relationship.

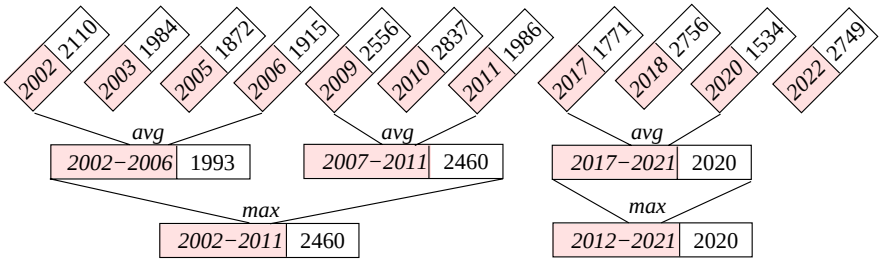


FIGURE 8. `tax_payment` attribute value

Note that Table 1 distinguishes between the insertion of a new value and the update of a value that is already present in the database, because each case requires a different re-computation. Finally, note that user-defined coercion functions do not appear in Table 1, because we do not know their operational semantics and thus we are not able *a priori* to establish whether a smart re-computation exists. If such smart re-computation is possible, the user should provide its procedural implementation.

EXAMPLE 5.3
Consider the class `tax_payer` of Example 2.2 and the value for attribute `tax_payments` in Figure 8. No value has been deleted from this attribute, but some values are missing, because for such years the tax declaration was not presented. Suppose we insert the new value $\{\langle$'2004', 2567$\rangle\}_{\,years}$ into the temporal value at granularity $years$. As a consequence, coarser temporal values must be updated. Then, the temporal value at granularity $5years$ with valid time '2002–2006' is re-computed applying the avg insertion formula of Table 1, which returns the temporal value $\{\langle$'2002–2006', 2088$\rangle\}_{\,5years}$. Moreover, the value at granularity $decades$ with valid time '2002–2011' is updated. Actually, the value remains unchanged.

By contrast, if we insert the value $\{\langle$'2009', 3000$\rangle\}_{\,years}$, we have to update the temporal value at granularity $years$ and also coarser level coherently, by applying the update formulas of Table 1 specified for coercion functions avg and max. Both the value at granularity $5years$ with valid time '2007–2011' and the value at granularity $decades$ with valid time '2002-2011' will be updated to 2608 value.

There are situations in which the effects of a nonmonotonic update should not be propagated. First of all, if we delete the oldest values from a representation level of an attribute, we require that this nonmonotonic change to the attribute value is not propagated to the coarser temporal values. This is in accordance with the semantics specified for deletion of values for dynamic attributes. Indeed, a value at a coarser level of detail can be kept even if the detailed value has been deleted.

Moreover, if we allow nonmonotonic updates on oldest data that have been deleted, in most cases we would not be able to propagate these updates in a meaningful way. Indeed, the re-application of the coercion function used to build the coarser value, or the propagation of the update according to formulas of Table 1, if possible, would corrupt it, because we would lose other information given from other deleted (and not re-inserted) values at the finer granularity, although present in a summarized form in the coarser value, as discussed above.

EXAMPLE 5.4
Consider the class `tax_payer` in Figure 7(a) and the `tax_payments` attribute whose values are reported in Figure 1. Values, referring valid years from 2002 to 2007, have been deleted as result of deletion operations.

Suppose now to insert the value $\{\langle$'2002',2500$\rangle\}_{years}$, that has been previously deleted. The propagation of this insertion to coarser levels results in a loss of information concerning years from 2003 to 2007. This is because, even if these values have been deleted from the temporal value at granularity $years$, they are still maintained, in a summarized form, at granularities $5years$ and $decades$. Indeed, the re-application of coercion functions avg and max should result in the updated temporal values $\{\langle$'2002–2006', 2500$\rangle, \langle$'2007–2011', 2303$\rangle\}_{5years}$, $\{\langle$'2002–2011', 2500$\rangle\}_{decades}$, and these values are not semantically correct.

Note that the few cases in which we can correctly perform the update propagation according to Table 1, are those in which we know at least what the value is, at finer level, to update. If this value has been deleted, the propagation is not possible any longer. Therefore, in $T\Omega$DMGe model nonmonotonic updates to values that have already been deleted are not allowed.

In the current version of the model *nonmonotonic updates for dynamic attributes are allowed also for values that are older than the evolution starting time at the base granularity level, and more recent than the update border*. With *update border* we denote a temporal bound that allows us to suspend the execution of deletions for a dynamic attribute until all nonmonotonic updates have been performed. When we are sure that no further nonmonotonic updates need to be performed for data referring to a particular temporal interval, we can set the update border to the greatest bound of this interval. Actually, the update border is not a time instant, rather it is a granule at the base granularity of the attribute.

EXAMPLE 5.5
Consider the class `tax_payer` in Figure 7(a). Suppose to set the update border for this attribute to granule '2001'. Then, we insert values for this attribute for years from 2003 to 2011, performing evolutions as described in Example 5.2, but not deletions of oldest values of granularity $years$. Then, suppose we perform the update of value with valid time '2002' of Example 5.4, followed by the update propagation to coarser levels that results in the temporal values $\{\langle$'2002–2006', 2168$\rangle, \langle$'2007–2011', 2303$\rangle\}_{5years}$, $\{\langle$'2002–2011', 2303$\rangle\}_{decades}$. Finally, we can set update border to granule '2007'. Then, the suspended deletions from temporal value at granularity $years$ of values from year 2002 until 2007 can be performed, without further propagations to coarser levels. Differently from Example 5.4, coarser levels maintain correctly updated information relative to years from 2002 until 2007, although in a summarized form.

Since the update border depends on the semantics of data and on the way they are inserted in the database, it is usually set referring to each particular attribute, by the user of the database having the knowledge to decide when no further nonmonotonic updates to each attribute will be performed.

The update border can also be incremented when a reduction of the database size is needed. To achieve the maximum database compression, the attribute update borders for all the objects in the database can be updated, possibly to the same instant. Such an operation must be considered with great care, since it implies that for all the dynamic attributes no further updates to data that precede the update border will be allowed by the system. Moreover, the update border must not be greater than the maximum starting time of the dynamic attribute in the database, in order to maintain the possibility of updating recent data that have not yet evolved.

5.3 Semantics of dynamic attribute updates

In this section we formalize the semantics of update operations for dynamic attributes. To explain how such an update is performed, we refer to Figure 9, in which we sketch the possible relationships between the evolution starting time of the base granularity level of a dynamic attribute and its update

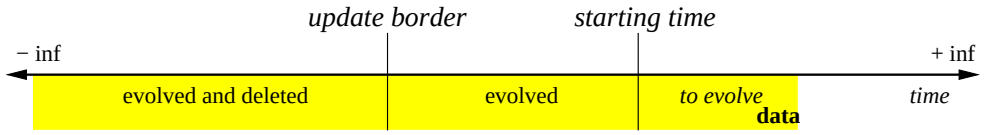


FIGURE 9. Relationships between starting time and update border of a dynamic attribute

border, with starting time that follows the update border. Note that the update border could be set to be more recent than starting time. Actually, we do not consider this case in the following discussion, because it is not really meaningful. Indeed, setting the update border before the starting time simply does not allow one to perform nonmonotonic updates to values for which no evolution has been performed yet, and these kinds of updates are not problematic at all, since they do not require recomputing of existing aggregations.

These time references partition data of a dynamic attribute into three groups. The first group includes data, older than or as old as the update border time, that are potentially deleted and for which some evolutions have already been performed. The second group includes data, more recent than the update border but older than the starting time, for which some granularity evolution have already been performed. Finally, the third group includes data whose valid time is more recent than or as recent as the starting time, that are represented only at the base attribute granularity, because no evolution has been performed yet for them.

When we update a value of a dynamic attribute, we specify its valid time by means of a granule of the base granularity. If the valid time of the updated value is older than the update border specified for the attribute, it is not allowed. By contrast, if the valid time of the updated value is more recent than the update border, the update is correct and then performed. In particular, if the valid time of the updated value falls in the temporal interval $]update\ border,\ starting\ time[$,[15] the update is an update of a value already present in the database or an insertion of a new value that needs to be propagated to attribute coarser levels. Thus, the update is performed according to the semantics of T_ODMG updates for temporal value, that is the value is simply updated or inserted into the temporal value at the base granularity, checking that it does not exceed the lifespan bounds of the object to which the attribute belongs. Then, the effects of this update are propagated to coarser levels according to the formulas of Table 1, or by re-applying coercion functions to the appropriate finer temporal value restrictions. If the valid time of the updated value falls in the temporal interval $[starting\ time,\ +\infty[$, the update surely refers to a value for which no evolutions have been performed yet, so it does not need to be propagated to any coarser level. After performing the update as in the previous case, we should determine if new expiration operations must be executed. To perform this check, we must consider the temporal interval bounded by the evolution and deletion starting times of the base granularity and the update valid time, to establish whether their duration include some expiration frequency, for expiration specified for this detail level. Then, if it is so, we must perform the expiration operations according to what we said in previous sections. In particular, values that can be deleted are only those whose valid times are older than update border. If some expiration has been performed, the corresponding starting time must be updated. In Figure 10(a) we report the operational semantics of update operation of dynamic attributes in pseudo code. In Figure 10(b) we also report the signatures of the various procedures used to specify the update semantics. In Figure 10(a) $upd\ border$ represents the attribute update border; $evolution\ st$ and $deletion\ st$ are the evolution and deletion starting times, respectively, of the temporal value at the base granularity;

[15] The open square brackets highlight that the bounds of the interval are not included in the interval itself.

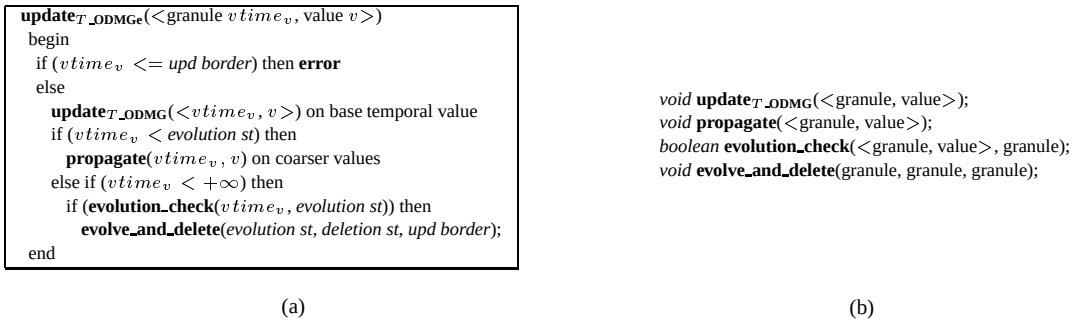


FIGURE 10. (a)Operational semantics of dynamic attribute updates. (b) Procedure signatures

$vtime_v$ is the valid time of the updated value; and v is the new value with valid time $vtime_v$.

5.4 Value deletion and object consistency

Value deletion can result apparently in instances that are not consistent according to Definition 2.5. If we delete values that have been involved in a granularity evolution, the intuitive requirement that the coarser temporal value is obtained simply by applying the corresponding coercion function to the immediately finer temporal value does not hold anymore. Actually, as we stressed before, if we requires that such a condition be satisfied for each state of a dynamic attribute, we do not obtain the correct semantics of value deletion that we want to achieve. Therefore Definition 2.5 does not impose any conditions on data that do not belong to the object state. Note that the case in which a value has never been inserted into the database, also not handled by Definition 2.5, is different from the case in which the same value has been deleted. In this case the consistency condition is verified anyway. As discussed in Section 5.1, we construct the coarser representation levels respecting the intuitive consistency condition that each level is obtained from the finer one through the application of the correct coercion function. In the same way, as discussed in the previous Section 5.3, we propagate the effects of updates to such data correctly. This is pointed out by the following Example.

EXAMPLE 5.6
Consider the class `tax_payer` of Example 2.2 and the attribute value for attribute `tax_payments` in Figure 8. Note that coarser levels are consistent with the application of coercion functions avg and max to the finer levels. Coarser attribute values were correctly constructed respecting the expiration process described in Section 5.1. Then, object consistency is achieved although the attribute value does not meet the conditions of Definition 2.5.

6 Related work

In this paper we address the problem of granularity evolution of temporal attributes and of deleting oldest portions of temporal values. Conditions for expiration 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 to data evolution and data expiration, separately. We remark, however, that our model differs from other approaches we refer to because the T_ODMGe model is an object-oriented data model, rather than a relational model.

6.1 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 [7, 12, 13, 18]. 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 from basic relations. By contrast, our approach allows the evolution of object attributes at increasing coarser levels of detail. Each evolution is computed by applying a coercion function on an attribute temporal value, and the result obtained 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. Moreover, [7, 12, 13, 18] consider transactional time as reference time dimension, whereas we consider valid time. 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 [20, 21], which, in turn, rely on the Chronicle Model [14]. In those approaches the concept of materialized *temporal views* is introduced. By relying on a materialized views concept, updates on source data are automatically propagated to materialized views. Moreover, temporal windows that slide as time advances and update to 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 value of a single attribute, thus, we have a different time window for each attribute of each object, instead of one for each class. Moreover, our model supports different time granularities and multiple levels of aggregation, while in [20, 21] a single time granularity and only one level of aggregation are considered. Finally, as we discussed in detail in Section 5, 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 detailed data are kept for one day, this means that they are aggregated when the first data referring to a new day are inserted. Under the sliding window approach of [20, 21] this means, by contrast, that in the middle of a day the detailed values of the last 24 hours are kept.

These differences have been weakened in [22], which rely on the approach presented in [20, 21], in which the *SB-Tree* indexing structure is proposed with several variations, to improve time response to queries that involve temporal aggregates. In particular, the time window approach that considers aggregates computed with respect to a fixed temporal window that slides over time, allows one to build, for the same attribute, several indexes, each one referring to a specific time window, that is, in turn, a granularity, and to a specific aggregate (sum, count, average, minimum and maximum aggregates are considered). However, the notion of granularity is still different from the one we use in our approach. The attribute values in [22] are timestamped at the same granularity for the whole database, and aggregates at different granularities are computed considering timestamp sets, without taking into account any semantics aspects related to sets of different cardinality (i.e. different granularities), then allowing one to compare these sets only in a punctual way (instant by instant). Moreover, the aggregates are computed referring to the entire set of values an attribute takes in a relational table, and not to a single attribute value or set of timestamps.

The indexing structure proposed in [22] has been extended in [23], which, together with the extension proposed in [24], addresses the same problem we deal with, and the solution of aggregated indexing proposed basically relies on the same idea we describe in this paper: older data are stored using granularities coaser than those used for recent data. With respect to our work, this approach is datawarehouse oriented, focusing on relational data and dealing with temporal granularities as a

datawarehouse conceptual hierachy. Finally, the indexing approach discussed in [24] involves only aggregations (in particularly sum, but the approach is easily extensible to count and average aggregation), and, in particular, all levels of detail are compressed using the same aggregation function. By contrast, our work relies on the standard notions of temporal granularity, as used by temporal database and reasoning communities, that considers granularities as data integrity constraints, and formalizes how different granularities are related to each other. We discuss different indexing forms, that is aggregated, selective and user defined indexing, and moreover different compression (i.e. coercion) functions can be used in the same time to index an attribute value, allowing one to express different semantics (i.e. different queries).

6.2 Deletion

The concept of *data expiration* was introduced in [11]. By relying on such a concept data can be removed (i.e. they *expire*) from the database without affecting related views. A similar approach was proposed for the time dimension in [19]. In [19] a technique is presented supporting automatic data deletion 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 of 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 is 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. Finally, our work relies on the same formal basis as [6], in which the data semantics, expressed by semantic assumptions [4], is used to derive a minimal rapresentation of information with respect to different granularities, that reduce the database size.

7 Conclusions and future work

In this paper we proposed T_ODMGe, a multigranular temporal object-oriented data model supporting the possibility of evolving data at coarser granularities, and of deleting or moving oldest data to tertiary storage devices, based on declaration, in the database schema, of expiration conditions for *dynamic attributes*. This approach has been implemented in the prototype implementation of T_ODMGe that has been developed using ObjectStore PSEPro for Java.

We are currently extending the work presented in the paper in 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.

One of the main limitations of the T_ODMGe model is that it deals with temporal attributes in isolation.[16] We are however extending the model with a general trigger language supporting attribute granularity evolution based on complex conditions, involving other attributes and other objects with conditions referring to the entire database. 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 expiration 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 the development of a bi-temporal data model. Finally, we are also investigating the problem of query evaluation in our multigranular model, as an

[16] This is in accordance with the *nonhomogeneity* characteristic of the T_ ODMGe model.

extension to the query evaluation mechanisms proposed for $T\mathcal{O}$DMG in [17, 1, 2].

Acknowledgements

A preliminary version of this paper appeared in *Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning*, 2002, with the title 'Evolution Specification of Multigranular Temporal Objects'.

References

[1] E. Bertino, E. Ferrari, G. Guerrini, and I. Merlo. Navigating through multiple temporal granularity objects. In *Proceedings of the Eighth International Symposium on Temporal Representation and Reasoning*, pp. 147–155, 2001.

[2] E. Bertino, E. Ferrari, G. Guerrini, I. Merlo. T-ODMG: an ODMG compliant temporal object model supporting multiple granularity management. In *Information Systems Journal*, **28**, 885–927, 2003.

[3] C. Bettini, C. Dyreson, W. Evans, R. Snodgrass, and X. Wang. A glossary of time granularity concepts. In *Proceedings of Temporal Databases: Research and Practice*, volume 1399 of *Lecture Notes in Computer Science*, pp. 406–413, Springer, 1998.

[4] C. Bettini, X.S Wang, and S. Jajodia. Temporal semantic assumptions and their use in databases. In *IEEE Transactions on Knowledge and Data Engineering*, **10**, 277–296, 1998.

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

[6] C. Bettini. Semantic compression of temporal data. In *Proceedings of the Second International Conference on Web-age Information Management*, volume 2118 of *Lecture Notes in Computer Science*, pp. 267–278. Springer, 2001.

[7] J. Blakeley, P. Larson, and F. Tompa. Efficiently updating materialized views. In *Proceedings of Special Interest Group on Management Of Data Conference*, pp. 61–71, 1986.

[8] E. Camossi, E. Bertino, G. Guerrini, and M. Mesiti. Evolution specification of multigranular temporal objects. In *Proceedings of the Ninth International Workshop on Temporal Representation and Reasoning*, pp. 78–85, 2002.

[9] R. Cattel, D. Barry, M. Berler, J. Eastman, D. Jordan, C. Russel, O. Schadow, T. Stanienda, and F. Velez. *The Object Database Standard: ODMG 3.0*. Morgan-Kaufmann, 1999.

[10] S. Gadia. A homogeneous relational model and query languages for temporal databases. In *ACM Transactions On Database Systems*, **14**, 418–448, 1988.

[11] H. Garcia-Molina, W. Labio, and J. Yang. Expiring data in a warehouse. In *Proceedings of the Twenty-fourth International Conference on Very Large Data Bases*, pp. 500–511, 1998.

[12] 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**, 3–18, 1995.

[13] N. Huyn. Efficient view self-maintenance. Technical Report, Stanford University, 1996.

[14] H. Jagadish, I. Mumick, and A. Silberschatz. View maintenance issues for the chronicle data model. In *Proceedings of the Fourteenth ACM Symposium on Principles of Database Systems*, pp. 113–124, 1995.

[15] C. Jensen and C. Dyreson. The consensus glossary of temporal database concepts. In *Proceedings of Temporal Databases: Research and Practice*, volume 1399 of *Lecture Notes in Computer Science*, pp. 366–405, Springer, 1998.

[16] I. Merlo. *Extending the ODMG Object Model with Temporal and Active Capabilities.* PhD thesis, Università di Genova, 2001.

[17] I. Merlo, E. Bertino, E. Ferrari, S. Gadia, and G. Guerrini. Querying multiple temporal granularity data. In *Proceedings of the Seventh International Workshop on Temporal Representation and Reasoning*, pp. 103–114, 2000.

[18] D. Srivastava, S. Dar, H. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proceedings of the Twenty-second International Conference on Very Large Data Bases*, pp. 318–329, 1996.

[19] D. Toman. Expiration of historical databases (extended abstract). In *Proceedings of the Eighth International Symposium on Temporal Representation and Reasoning*, 2001.

[20] J. Yang and J. Widom. Maintaining temporal views over non-temporal information sources for data warehousing. In *Proceedings of the Sixth International Conference on Extending Database Technology*, pp. 389–403, 1998.

[21] J. Yang and J. Widom. Temporal view self-maintenance. In *Proceedings of the Seventh International Conference on Extending Database Technology*, pp. 395–412, 2000.

[22] J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. In *Proceedings of the Seventeenth International Conference on Data Engineering*, pp. 51–60, 2001.

[23] D. Zhang, D. Gunopulos, V. J. Tsotras, and B. Seeger. Temporal aggregation over data streams using multiple granularities. In *Proceedings of the Eighth International Conference on Extending Database Technology*, pp. 646–663, 2002.

[24] D. Zhang, D. Gunopulos, V. J. Tsotras, and B. Seeger. Temporal and spatio-temporal aggregation over data streams using multiple time granularities. In *Information Systems Journal*, **28**, 61–84, 2003.

Received 22 October 2002