# A Temporal Object-Oriented Data Model with Multiple Granularities

Isabella Merlo[1] Elisa Bertino[2] Elena Ferrari[2] Giovanna Guerrini[1]

[1]Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova - Italy
{merloisa, guerrini}@disi.unige.it

[2]Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano - Italy
{bertino, ferrarie}@dsi.unimi.it

## Abstract

*In this paper we investigate some issues arising from the introduction of multiple temporal granularities in an object-oriented data model. Although issues concerning temporal granularities have been investigated in the context of temporal relational database systems, no comparable amount of work has been done in the context of object-oriented models. Moreover, the main drawback of the existing proposals is the lack of a formal basis – which we believe is essential to manage the inherent complexity of the object-oriented data model. In this paper, we provide a complete temporal object-oriented type system supporting multiple temporal granularities and we formally define the set of legal values for our type system. We then address issues related to inheritance, type refinement and substitutability.*

## 1. Introduction

Conventional database systems do not offer the possibility of dealing with time-varying data. The content of a database represents a *snapshot* of the reality in that only the current data are recorded, without the possibility of maintaining the complete history of data over time. If such a need arises, data histories must be managed at application program level, thus, making the management of data very difficult, if at all possible. To overcome this lack, in the past years, there has been a growing interest in temporal extensions to the current database technology. Several extensions to the relational and the object-oriented data models and the associated query languages have been proposed [13].

An important requirement, when dealing with temporal aspects, concerns the support for *multiple temporal granularities*. Multiple temporal granularities allow one to store temporal data according to different temporal units, depending on the needs of the application domain. The choice of the correct temporal granularity not only allows the system to store the minimal amount of data, but also establishes an important integrity constraint: data cannot be changed more often than the specified granularity.

The relevance of multiple granularities in the context of temporal databases is well recognized by the scientific community. A considerable amount of work has been developed in the context of temporal relational database systems [4, 5]. For instance, the standard for temporal relational database systems, TSQL-2 [11], supports multiple granularities, whereas in [12], temporal granularities are used in the context of temporal relational databases to express constraints over data. Such constraints are modeled by temporal functional dependencies.

By contrast, no comparable amount of work has been carried out in the context of the object-oriented data model, where the introduction of multiple temporal granularities poses additional issues with respect to the relational context, due to the semantic richness of such model. Most of the temporal object-oriented data models proposed so far do not deal with temporal granularities. The only ones dealing with temporal granularities [8, 9, 10, 14] support multiple temporal granularities as extensions to the set of types of the temporal model. However, in most of these approaches the specification and the management of these different granularities, e.g. how to convert from a granularity to another, is completely left to the user. Moreover, in contrast to the relational context, the introduction of temporal granularities in object-oriented data models is, in most cases, informal. For instance, none of the proposed approaches consider the impact of inheritance when multiple granularities are supported, nor they address issues concerning consistency, type refinement and substitutability. We believe that these are important topics to be investigated and that their treatment on a formal basis is a means to manage their complexity.

In this paper we make a step in this direction by proposing a complete temporal type system supporting multiple temporal granularities. First, we formally define the notion of temporal type with respect to a given granularity, then we provide a formal definition for the set of legal values of our temporal type system. Finally, the notions of inheritance, type refinement and substitutability are revisited taking multiple granularities into account. Substitutability ensures that each instance of a given class can be used whenever an instance of one of its superclasses is expected. To enforce substitutability, the idea is that attribute domains can be replaced in a subclass by refined types. For instance, an attribute with granularity month can be refined in an attribute with granularity day defined on the same domain or on a more specific one. This means that in the subclass the history of the daily changes of the attribute is kept. Clearly the attribute may not maintain the same value for each day of the month. Then, the question arises of which value should be considered when the value of the attribute in a given month is required, that is, when an instance of the subclass must be seen as an instance of the superclass. To overcome this problem, we introduce the notion of *coercion function*. Coercion functions are used to compute, given the values of the attribute in the subclass, the value to be considered in the superclass.

The remainder of this paper is organized as follows. Section 2 presents the object model we refer to in the paper and the notion of temporal granularity. Section 3 formally introduces our temporal type system, whereas issues related to inheritance are discussed in Section 4. In Section 5 an illustrative example is presented. Finally, Section 6 concludes the paper and outlines future research directions.

## 2. Preliminaries

In this section we first characterize the reference object model we refer to in the paper, then we introduce some preliminary concepts related to multiple temporal granularities.

### 2.1. Reference Object Model

In the reference object model we assume the existence of two kinds of types: *object* types and *literal* types, as in the ODMG data model [7]. The main difference between object and literal types is that values belonging to object types, defined through classes, have object identifiers, whereas values belonging to literal types are identified by themselves. Moreover, for both object and literal types we assume the existence of a set of predefined basic types and of a set of constructors of collection types. Examples of collection types are sets, lists, arrays and so on. The previous notions are formalized by the following definitions.

**Definition 1** (*Object Types*). The set of object types, $\mathcal{OT}$, is recursively defined as follows:

- the predefined object types, $\mathcal{BOT}$, are object types ($\mathcal{BOT} \subseteq \mathcal{OT}$);
- the class identifiers, $\mathcal{CI}$, are object types ($\mathcal{CI} \subseteq \mathcal{OT}$);
- let $\mathtt{o\_constr}$ be a general type constructor for collection object types, then, for each type $\mathtt{t}$, $\mathtt{o\_constr<t>}$ is a collection object type ($\mathtt{o\_constr} < \mathtt{t} > \in \mathcal{OT}$).

□

**Definition 2** (*Literal Types*). The set of literal types, $\mathcal{LT}$, is recursively defined as follows:

- the predefined literal types, $\mathcal{BLT}$, are literal types ($\mathcal{BLT} \subseteq \mathcal{LT}$);
- let $\mathtt{l\_constr}$ be a general type constructor for collection literal types, then, for each type $\mathtt{t}$, $\mathtt{l\_constr<t>}$ is a collection literal type ($\mathtt{l\_constr} < \mathtt{t} > \in \mathcal{LT}$);
- let $\mathtt{a}_1, ..., \mathtt{a}_n$ be distinct labels and $\mathtt{t}_1, ..., \mathtt{t}_n$ be types, then $\mathtt{struct} < \mathtt{t}_1\mathtt{a}_1, ..., \mathtt{t}_n\mathtt{a}_n >$ is a struct literal type ($\mathtt{struct} < \mathtt{t}_1\mathtt{a}_1, ..., \mathtt{t}_n\mathtt{a}_n > \in \mathcal{LT}$). □

Note that our model provides both collection object types and collection literal types. The difference between the two is that values of collection object types have an identifier, whereas values of collection literal types do not have an identifier.

The set of types we refer to throughout the paper, denoted as $\mathcal{ST}$,[1] is defined as the union of the set of object types and the set of literal types, that is $\mathcal{ST} = \mathcal{OT} \cup \mathcal{LT}$.

As far as literal types are concerned, given a type $\mathtt{t}$ belonging to $\mathcal{LT}$, we denote with $dom(\mathtt{t})$ the set of values of type $\mathtt{t}$. By contrast, if $\mathtt{t}$ belongs to $\mathcal{OT}$, then the set of values of type $\mathtt{t}$ is the set of object identifiers belonging to $\mathtt{t}$. Since objects can be dynamically created and deleted, the set of values of an object type depends on time.[2] We do not explicitly specify the set of legal values corresponding to each type of the reference model since they are not relevant for the remainder of the discussion. Details on these aspects can be found in [2]. Legal values for the extended temporal model are presented in subsection 3.2.

As far as the class definition is concerned, our notion of class is one of the most common definition that can be found

[1]$\mathcal{ST}$ stands for static types, since these types do not include the time dimension.

[2]These concepts are formalized in subsection 3.2.

in the literature [1]. In our model classes are characterized by an *identifier*, $c \in \mathcal{CI}$, which represents the type of the objects belonging to the class, a set of *attributes* and a set of *method* signatures. The attributes of a class are characterized by a name and a type, which represents the set of legal values for that attribute, that is, the attribute *domain*. Method signatures are characterized by a method name, the name and type of input parameters and the name and type of output parameters. We do not provide here a formal definition of classes, since it is not relevant for the remainder of the discussion.

Classes, or equivalently, object types, are organized into an inheritance hierarchy, named *ISA* hierarchy, whose properties and features will be described in Section 4. The following example illustrates the notation we use throughout the paper to represent classes.

**Example 1** Let `Person` be a given class. Moreover, let `Employee` be a class, subclass of `Person` with proper attributes `emp_nbr` and `name` of type `string`, `age` and `salary` of type `integer` and `participates` of type `Project`. Let `increase_salary` be a method that given an amount `amount` increases the salary of the employee of the given amount. A possible declaration for class `Employee` is the following:

```
Employee {
  emp_nbr:  string;
  name:  string;
  age:  integer;
  salary:  integer;
  participates:  Project;

  increase_salary(integer amount);
  }
```

◇

## 2.2. Multiple Granularities

Issues concerning temporal granularities are a recent research topic in the temporal database area. A glossary of time granularity concepts [3] has been recently published. In [3] all the main concepts concerning temporal granularities are formally defined without referring to any particular data model. To be as general as possible in extending the object model presented in the previous section with temporal granularities, we refer to the definitions in [3].

As suggested in the glossary, one of the first notion to be fixed in a temporal context is the *time domain*, that is, the set of primitive temporal entities used to define and interpret time-related concepts. To model the time dimension, we introduce a new type `time` whose set of values is the set of natural numbers $\mathbb{N}$ (that is, *dom*(`time`) = $\mathbb{N}$). Thus, we assume the existence of a relative beginning denoted by symbol "0", but no last element. Moreover, we consider a special variable $now$ denoting the current time. Therefore, in our model, the time domain is the pair ($\mathbb{N}$,$\leq$), where $\leq$ is the order on $\mathbb{N}$. We recall that our time domain is discrete since every element has an immediate successor, and every element except the first has an immediate predecessor. Temporal granularities are formally defined as follows.

**Definition 3** (*Granularity*)[3]. Let $\mathcal{IS}$ be an index set, that is a set of integers, and $2^{\mathbb{N}}$ be the power set of the time domain. A *granularity* $G$ is a mapping from $\mathcal{IS}$ to $2^{\mathbb{N}}$ such that both the following conditions hold:

(1) if $i < j$ and $G(i)$ and $G(j)$ are non-empty, then each element of $G(i)$ is less than all elements of $G(j)$;

(2) if $i < k < j$ and $G(i)$ and $G(j)$ are non-empty, then $G(k)$ is non-empty. □

Intuitively a granularity defines a countable set of granules, each granule $G(i)$ identified by an integer. The first condition in Definition 3 states that granules in a granularity do not overlap and that their index order is the same as their time domain order. The second condition states that the subset of the index set that maps to non-empty subsets of the time domain is contiguous. The usual collections $days$, $months$, $weeks$ and $years$ are granularities. For readability, we use a "textual representation" for each non-empty granule, termed as *label*, which is more descriptive than the granule index. For example, throughout the paper, $days$ are in the form $mm/dd/yy$, $months$ are in the form $mm/yy$ and so on.

A *finer than* relationship can be defined among temporal granularities [3], with the following meaning. A granularity $G$ is said to be *finer than* a granularity $H$, denoted $G \preceq H$, if for each index $i$, there exists an index $j$ such that $G(i) \subseteq H(j)$. For example, $days$ is a granularity finer than $months$ ($days \preceq months$). In the following the finer than relationship will be used to establish subtyping relationships among temporal types and to state how attribute domains can be refined along the inheritance hierarchy.

## 3. Temporal Type System

In this section, the type system described in subsection 2.1 is extended by adding temporal types related to different granularities. Furthermore, the notion of set of values of a given type is formally defined according to the new set of temporal types.

### 3.1. Temporal Types and Multiple Granularities

We extend the set of types of the model, $\mathcal{ST}$, with a collection of temporal types, so that temporal and non-temporal domains can be handled in a uniform way. First, the set of basic literal types is extended with the type `time`. Then, for each type $\mathtt{t} \in \mathcal{ST}$ and granularity $G$, a corresponding temporal type, $temporal_G(\mathtt{t})$, is defined. Intuitively, instances of type $temporal_G(\mathtt{t})$ are partial functions from granules of $G$ to instances of type $\mathtt{t}$.[3]

**Definition 4** (*Temporal Types*). Let $\mathtt{t} \in \mathcal{ST}$ be a type and $G$ be a temporal granularity, then $temporal_G(\mathtt{t})$ is the temporal type corresponding to type $\mathtt{t}$ and granularity $G$. □

In the following the set of temporal types will be denoted as $\mathcal{TT}$, whereas $\mathcal{T}$ denotes the whole set of the types provided by our model.

Temporal types, defined with respect to a given granularity, are particularly meaningful in the context of databases. If an attribute $a$ of an object $o$ has type $temporal_G(\mathtt{t})$, then such attribute cannot vary more than once for each granule of $G$. Therefore, given a granule $G(i)$, the value of the attribute is the same for each instant $t \in G(i)$. In our model, temporal types can be used in the definition of collection object and literal types, as stated by the following definitions.

**Definition 5** (*Object Types*). The set of object types, $\mathcal{OTT}$, is recursively defined as follows:

- each element of $\mathcal{OT}$ (see Definition 1) belongs to $\mathcal{OTT}$ ($\mathcal{OT} \subseteq \mathcal{OTT}$);
- for each type $\mathtt{t} \in \mathcal{T}$, `o_constr<t>` (see Definition 1) is a collection object type (`o_constr<t>` $\in \mathcal{OTT}$).

□

**Definition 6** (*Literal Types*). The set of literal types, $\mathcal{LTT}$, is recursively defined as follows:

- `time` $\in \mathcal{LTT}$;
- each element of $\mathcal{LT}$ (see Definition 2) belongs to $\mathcal{LTT}$ ($\mathcal{LT} \subseteq \mathcal{LTT}$);
- for each type $\mathtt{t} \in \mathcal{T}$, `l_constr<t>` (see Definition 2) is a collection literal type (`l_constr<t>` $\in \mathcal{LTT}$);
- let $\mathtt{a}_1, \ldots, \mathtt{a}_n$ be distinct labels and $\mathtt{t}_1, \ldots, \mathtt{t}_n$ be types in $\mathcal{T}$, then $\mathtt{struct}\langle\mathtt{t}_1\ \mathtt{a}_1, \ldots, \mathtt{t}_n\ \mathtt{a}_n\rangle$ is a struct literal type ($\mathtt{struct}\langle\mathtt{t}_1\ \mathtt{a}_1, \ldots, \mathtt{t}_n\ \mathtt{a}_n\rangle \in \mathcal{LTT}$). □

The set of types of our temporal model is defined as the union of object, literal and temporal types, that is, $\mathcal{T} = \mathcal{OTT} \cup \mathcal{LTT} \cup \mathcal{TT}$.

[3]We elaborate on that throughout the paper.

## 3.2. Temporal Values

In this section we define the set of legal values supported by our model. Let $OID$ be the set of possible object identifiers. We define a function $\pi : \mathcal{OTT} \times \mathbb{N} \rightarrow 2^{OID}$, that, given an object type $\mathtt{t}$ and a time instant $t$, returns the set of identifiers of objects belonging to type $\mathtt{t}$ at time $t$. We define the set of legal values of each type $\mathtt{t} \in \mathcal{T}$ by using three functions, namely $Eval$, $Eval_G$ and $T\_Eval$, formally defined as follows.

**Definition 7** (*Legal Values of a Non-Temporal Type with Respect to the Time Domain*). Let $\mathtt{t} \in \mathcal{T} \setminus \mathcal{TT}$ be a non-temporal type and $t \in \mathbb{N}$ be a time instant, then $Eval(\mathtt{t}, t)$ denotes the extension of type $\mathtt{t}$ at time $t$:

$$Eval(\mathtt{t}, t) = \begin{cases} dom(\mathtt{t}) & \text{if } \mathtt{t} \in \mathcal{LTT} \\ \pi(\mathtt{t}, t) & \text{if } \mathtt{t} \in \mathcal{OTT} \end{cases}$$ □

Note that, according to the previous definition, only object types have extents which depend on time. Intuitively, the set of values of a literal type does not change over time since no literal values can be explicitly created or deleted, whereas objects belonging to object types are dynamically created and deleted, thus an object type extent depends on the considered instant.

When dealing with temporal granularities, function $Eval$ is generalized to a granularity $G$ through the following definition.

**Definition 8** (*Legal Values of a Non-Temporal Type with Respect to a Granule*). Let $\mathtt{t} \in \mathcal{T} \setminus \mathcal{TT}$ be a non-temporal type, $G$ be a granularity and $i \in \mathcal{IS}$ be an index, then $Eval_G(\mathtt{t}, i)$ denotes the extension of type $\mathtt{t}$ with respect to the granule identified by $i$:

$$Eval_G(\mathtt{t}, i) = \bigcap_{t \in G(i)} Eval(\mathtt{t}, t)$$

where $G(i)$ is the set of time instants corresponding to granule $i$ with respect to granularity $G$. □

**Example 2** Let `department` be a class such that $Eval_{years}(\mathtt{department},\ 1997)$[4] $= \{\mathtt{d}_1, \mathtt{d}_2,\ \mathtt{d}_3\}$. Then, for each instant belonging to 1997, $\mathtt{d}_1, \mathtt{d}_2,\ \mathtt{d}_3$ must exist. Formally, $Eval_{years}(\mathtt{department},\ 1997) = \bigcap_{t \in I} Eval(\mathtt{department}, t)$ where $I$ denotes the set of instants corresponding to 1997, $years(1997) = I \subseteq \mathbb{N}$. The idea is that if a department exists only during a portion of a year it does not belong to the extent of class `department` of that year. ◇

We are now ready to define the set of legal values for temporal types with respect to a given time instant $t \in \mathbb{N}$.

[4]For simplicity, the label $1997$ is used instead of the index $i$ corresponding to such granule.

**Definition 9** (*Temporal Type Legal Values*). Let $temporal_G(\mathtt{t}) \in \mathcal{TT}$ be a temporal type and $t \in \mathbb{N}$ be a time instant, then $T\_Eval(temporal_G(\mathtt{t}), t)$ denotes the set of legal values of type $temporal_G(\mathtt{t})$ at instant $t$:

$$\begin{aligned} &T\_Eval(temporal_G(\mathtt{t}), t) = \\ &\quad \{f \mid f = \bar{f} \circ G \text{ such that } \bar{f} : 2^{\mathbb{N}} \to \textstyle\bigcup_{i \in \mathcal{IS}} Eval_G(\mathtt{t}, i) \\ &\quad \text{is a partial function such that for each } i \in \mathcal{IS} \\ &\quad \text{if } \bar{f}(G(i)) \text{ is defined then } \bar{f}(G(i)) \in Eval_G(\mathtt{t}, i)\} \end{aligned}$$

□

Note that the set of legal values of a given temporal type does not depend on the particular time instant. Formally we can state that:

$$\textstyle\bigcup_{t \in \mathbb{N}} T\_Eval(temporal_G(\mathtt{t}), t) = T\_Eval(temporal_G(\mathtt{t}), \bar{t}) \text{ for each } \bar{t} \in \mathbb{N}$$

Thus in the following we will omit the time instant argument. Since the set of values of a given temporal type does not depend on time, we can extend function $Eval_G$ (Definition 8) to temporal types as follows:

$$\begin{aligned} &Eval_G(temporal_H(\mathtt{t}), i) = \\ &\textstyle\bigcap_{t \in G(i)} T\_Eval(temporal_H(\mathtt{t}), t) = \\ &T\_Eval(temporal_H(\mathtt{t}), t) \end{aligned}$$

The following example clarifies the above definitions.

**Example 3** Let `department` be a class such that: $D = \bigcup_{i \in \mathcal{IS}} Eval_{years}(\mathtt{department}, i) = \{\mathtt{d}_1, \mathtt{d}_2, \mathtt{d}_3, \ldots, \mathtt{d}_n\}$, then, according to Definition 9:

$$\begin{aligned} &T\_Eval(temporal_{years}(\mathtt{department}), t) = \\ &\quad \{f \mid f = \bar{f} \circ years \text{ such that } \bar{f} : 2^{\mathbb{N}} \to D \\ &\quad \text{is a partial function such that for each} \\ &\quad i \in \mathcal{IS} \text{ if } \bar{f}(years(i)) \text{ is defined then} \\ &\quad \bar{f}(years(i)) \in Eval_{years}(\mathtt{t}, i)\} \end{aligned}$$

Examples of functions, denoted as set of couples, in $T\_Eval(temporal_{years}(\mathtt{department}), t)$ are:

$$\{\langle I_{1992}, \mathtt{d}_1\rangle, \langle I_{1993}, \mathtt{d}_4\rangle\};$$

$$\{\langle I_{1995}, \mathtt{d}_2\rangle, \langle I_{1998}, \mathtt{d}_2\rangle\}$$

where $years(1992) = I_{1992}$, $years(1993) = I_{1993}$ and so on. ◇

## 4. Inheritance

Inheritance relationships among object types are described by an ISA hierarchy established by the user. The ISA hierarchy represents which classes are subclasses of (inherit from) other classes. This information is expressed as a partial order $<_{ISA}$ on the set of object types $\mathcal{OTT}$. The reflexive closure of the $<_{ISA}$ relation will be denoted as $\leq_{ISA}$. Starting from the ISA hierarchy, a relation $\leq_T$ can be imposed on the set of types $\mathcal{T}$.

**Definition 10** (*Subtypes*). Let $\mathtt{t}_1, \mathtt{t}_2 \in \mathcal{T}$ be two types, and let $G$ and $H$ be two temporal granularities, then, $\mathtt{t}_2$ is a subtype of $\mathtt{t}_1$ (denoted as $\mathtt{t}_2 \leq_T \mathtt{t}_1$) iff one of the following conditions holds:

- $\mathtt{t}_1, \mathtt{t}_2 \in \mathcal{BOT} \cup \mathcal{CI}$ and $\mathtt{t}_2 \leq_{ISA} \mathtt{t}_1$;
- $\mathtt{t}_1$ = `o_constr<`$\mathtt{t}'_1$`>`, $\mathtt{t}_2$ = `o_constr<`$\mathtt{t}'_2$`>` and $\mathtt{t}'_2 \leq_T \mathtt{t}'_1$;
- $\mathtt{t}_1$ = `l_constr<`$\mathtt{t}'_1$`>`, $\mathtt{t}_2$ = `l_constr<`$\mathtt{t}'_2$`>` and $\mathtt{t}'_2 \leq_T \mathtt{t}'_1$;
- $\mathtt{t}_1$ =`struct<`$\mathtt{t}'_1\ \mathtt{a}_1, \ldots, \mathtt{t}'_n\ \mathtt{a}_n$`>`, $\mathtt{t}_2$ =`struct<`$\mathtt{t}''_1\ \mathtt{a}_1, \ldots, \mathtt{t}''_n\ \mathtt{a}_n$`>` and $\forall i \in [1, n]$: $\mathtt{t}''_i \leq_T \mathtt{t}'_i$;
- $\mathtt{t}_1 = temporal_G(\mathtt{t}'_1)$, $\mathtt{t}_2 = temporal_H(\mathtt{t}'_2)$, $\mathtt{t}'_2 \leq_T \mathtt{t}'_1$ and $G \preceq H$. □

Subtyping has two important implications in object-oriented models [6]. The first is *extent inclusion*, that is, the property ensuring that the extent of a subtype is included in the extent of its supertype. The second is $substitutability$, that is, the property that each instance of a type can be used whenever an instance of one of its supertypes is expected. The following proposition ensures that $\leq_T$ satisfies the extent inclusion property.

**Proposition 1** *Let* $\mathtt{t}_1, \mathtt{t}_2 \in \mathcal{T}$ *be two types such that* $\mathtt{t}_2 \leq_T \mathtt{t}_1$*, then the set of values of type* $\mathtt{t}_2$ *is a subset of the set of values of type* $\mathtt{t}_1$. △

**Proof Sketch** Extent inclusion can be easily proved for non-temporal types. For example, let $c_1$ and $c_2$ be two classes such that $c_2 \leq_T c_1$. It is obvious that the values of type `set<`$c_2$`>` are a subset of the values of type `set<`$c_1$`>`. We focus our attention on temporal types. Let $\mathtt{t}_2 = temporal_H(\mathtt{t}'_2)$ and $\mathtt{t}_1 = temporal_G(\mathtt{t}'_1)$. To prove the proposition we should prove that the set of values of type $temporal_H(\mathtt{t}'_2)$ is a subset of the set of values of type $temporal_G(\mathtt{t}'_1)$. Formally, this means to prove that $T\_Eval(temporal_H(\mathtt{t}'_2)) \subseteq T\_Eval(temporal_G(\mathtt{t}'_1))$. According to Definition 9:

$$\begin{aligned} &H_{set} = T\_Eval(temporal_H(\mathtt{t}'_2)) = \\ &\{f \mid f = \bar{f} \circ H \text{ such that } \bar{f} : 2^{\mathbb{N}} \to \textstyle\bigcup_{i \in \mathcal{IS}} Eval_H(\mathtt{t}'_2, i) \\ &\quad \text{is a partial function such that for each } i \in \mathcal{IS} \\ &\quad \text{if } \bar{f}(H(i)) \text{ is defined then } \bar{f}(H(i)) \in Eval_H(\mathtt{t}'_2, i)\} \end{aligned}$$

and

$$\begin{aligned} &G_{set} = T\_Eval(temporal_G(\mathtt{t}'_1)) = \\ &\{f' \mid f = \bar{f}' \circ G \text{ such that } \bar{f}' : 2^{\mathbb{N}} \to \textstyle\bigcup_{i \in \mathcal{IS}} Eval_G(\mathtt{t}'_1, i) \\ &\quad \text{is a partial function such that for each } i \in \mathcal{IS} \\ &\quad \text{if } \bar{f}'(G(i)) \text{ is defined then } \bar{f}'(G(i)) \in Eval_G(\mathtt{t}'_1, i)\} \end{aligned}$$

Since $G \preceq H$, for each index $i$, an index $j$ exists such that $G(i) \subseteq H(j)$. This implies that for each function $f = \bar{f} \circ H \in H_{set}$, a function $f' = \bar{f}' \circ G \in G_{set}$ exists such that if $\bar{f}(H(j)) = v$, then $\bar{f}'(G(i)) = v$ for each $i$ such that $G(i) \subseteq H(j)$. $\bigcirc$

The extent inclusion property is automatically satisfied by two classes related by the ISA relationship, whereas to ensure substitutability, some conditions must be satisfied. Those conditions are related to the fact that each subclass must contain all attributes and methods of its superclasses. Apart from the inherited features, additional features can be introduced in a subclass. Inherited features may be redefined (overwritten) in a subclass under a number of restrictions. In our model the redefinition of an attribute domain is possible by specializing the domain of the attribute. In this paper, we do not deal with method refinement, but our approach can be easily extended to the refinement of method signatures.

The idea is that attribute domains can be replaced in subclasses by *refined types*. A type $\mathtt{t}_2$ is a refinement of a type $\mathtt{t}_1$ if $\mathtt{t}_2$ is subtype of $\mathtt{t}_1$ ($\mathtt{t}_2 \leq_T \mathtt{t}_1$). In the context of temporal types, this means, for example, that an attribute defined with granularity $months$ can be refined in an attribute with granularity $years$ on the same domain or on a most specific one, since the temporal type of the attribute in the superclass is a supertype of the temporal type of the attribute in the subclass.

However, we would like also to allow an attribute defined with a given granularity to be refined in an attribute with a finer granularity on the same domain or on a most specific one. An example is to refine an attribute defined with granularty $years$ in an attribute with granularity $months$. The reason is that, according to our point of view, attribute refinement is a meaningful way of adding information in the refined attribute. More precisely, by refining the granularity associated with an attribute, one can specify more detailed information for a given time interval. For example, if an attribute $\mathtt{a}$ defined with granularity $years$ is specialized in a subclass in an attribute with granularity $months$, then more information for that attribute can be stored in the subclass. In the superclass, only values for each year can be stored for $\mathtt{a}$, whereas in the subclass values for each mounth of each year can be stored for $\mathtt{a}$.

However, to ensure substitutability, in case of attribute refinement where the refining attribute domain has a finer granularity with respect to the refined attribute domain, we need to introduce a *coercion* function to compute the value of the refined attribute with respect to the coarser granularity. The following example clarifies these concepts.

**Example 4** Let `Person` and `Employee` be two classes such that $\mathtt{Employee} \leq_{ISA} \mathtt{Person}$. Moreover, let $\mathtt{a}$ be a temporal attribute whose domain is $temporal_G(\mathtt{t})$ in `Person` and whose refined domain in `Employee` is $temporal_H(\mathtt{t})$. For simplicity, we assume the inner type $\mathtt{t}$ to be the same, thus we can focus on temporal granularities. We have two cases:

Case 1. $G \preceq H$, for example $G = months$ and $H = years$:

$\mathtt{Person}\{\ldots\mathtt{a}{:}temporal_{months}(\mathtt{t});\ldots\}$
$\mathtt{Employee}\{\ldots\mathtt{a}{:}temporal_{years}(\mathtt{t});\ldots\}$

In this case, since $temporal_{years}(\mathtt{t})$ is a subtype of $temporal_{months}(\mathtt{t})$ no problems arise. When an object of type `Person` is accessed we expect the type of attribute $\mathtt{a}$ to be $temporal_{months}(\mathtt{t})$, but if such object is an instance of `Employee` we will have an object whose attribute $\mathtt{a}$ is of type $temporal_{years}(\mathtt{t})$. However, if attribute $\mathtt{a}$ has a value for a given year, such value is the same for each month of the year. Thus, we do not need any additional information to map one granularity onto the other while accessing objects of type `Person`.

Case 2. $H \preceq G$, for example $H = months$ and $G = years$:

$\mathtt{Person}\{\ldots\mathtt{a}{:}temporal_{years}(\mathtt{t});\ldots\}$
$\mathtt{Employee}\{\ldots\mathtt{a}{:}temporal_{months}(\mathtt{t});\ldots\}$

In this case for an instance of type `Employee` we may know the value of the attribute just in some months of the year, but not in all the months of the year. Thus, the problem is to determine which value among the ones in which the attribute is defined must be considered for the coarser granularity. Suppose for example that one is interested in knowing the value of attribute $\mathtt{a}$ corresponding to year `1994`. Then, all the values corresponding to the months of such year will be retrieved and the value related to year `1994` will be computed according to a *coercion function*. $\diamond$

Our idea is to add information to the refined attribute which allows one to choose the value. The general idea is that if an attribute of type $temporal_G(\mathtt{t}'_1)$ is refined in a subclass into an attribute of type $temporal_H(\mathtt{t}'_2)$, with $\mathtt{t}'_2 \leq_T \mathtt{t}'_1$ and $H \preceq G$, a *coercion function* must be defined for the attribute, according to which the attribute value is computed. When the attribute is accessed, with respect to a specific granule $j$ of granularity $G$, the coercion function is applied to all the granules $i_1, \ldots, i_n$ such that $H(i_k) \subseteq G(j)$, $k \in [1, n]$. The following definition formalizes the notion of coercion function.

**Definition 11** (*Coercion Function*).
Let $\mathtt{t}_1 = temporal_G(\mathtt{t}'_1)$ and $\mathtt{t}_2 = temporal_H(\mathtt{t}'_2)$ be two temporal types such that, $\mathtt{t}'_2 \leq_T \mathtt{t}'_1$ and $H \preceq G$. A coercion function $C$ is a partial function defined as:

$$C : T\_Eval(temporal_H(\mathtt{t}'_2)) \rightarrow T\_Eval(temporal_G(\mathtt{t}'_1))$$

that maps values of type $temporal_H(\mathtt{t}'_2)$ into values of type $temporal_G(\mathtt{t}'_1)$. □

A large variety of coercion functions could be devised. We have developed a simple language for defining coercion functions. In our approach, a coercion function is associated with an attribute definition, since such function establishes how values of the attributes have to be coerced to the coarser granularity. The syntax in BNF form of attribute specifications, whose semantics will be clarified in the remainder of the section, is given in Figure 1. With reference to Figure 1 terminal symbol `index` denotes an element in $\mathcal{IS}$, `meth_inv` denotes a method invocation and square brackets denote optional symbols. As specified in the BNF grammar of Figure 1, depending on how the value of a granule $G(j)$ is computed with respect to the values of the granules $H(i)$, such that $H(i) \subseteq G(j)$, coercion functions can be classified into three categories: *selective*, *aggregate* and *user-defined* coercion functions, whose meaning is formally defined in Definition 12. In Figure 1 terminal symbols `first`, `last` and `Proj(index)` denote selective coercion functions of obvious meaning, whereas `min`, `max`, `avg` and `sum` denote the well-known SQL aggregate functions.

**Definition 12** (*Coercion Function Classification*). A coercion function $C$ can be classified as follows.

- Let $\{i_1, \ldots, i_k\}$ be the set of indexes such that $H(i_k) \subseteq G(j)$ and $f(H(i_k)) = v_k$, then $C$ is a *selective* coercion function if $C(f) = f'$ where $f'(G(j)) = v_{\bar{k}}$, $\bar{k} \in [1, n]$.
- Let $\{i_1, \ldots, i_k\}$ be the set of indexes such that $H(i_k) \subseteq G(j)$ and $f(H(i_k)) = v_k$, then $C$ is an *aggregate* coercion function if $C(f) = f'$ where $f'(G(j)) = \bar{v}$ and $v$ is computed as function of all the previous values.
- Let $f$ be a value of type $temporal_H(\mathtt{t}'_2$, then a *user-defined* coercion function $C$ is such that the value $f'$ of type $temporal_G(\mathtt{t}'_1$ to be returned by function $C$ is a user-defined method. □

Intuitively, in case of selective coercion functions, one of the possible values among $\{v_1, \ldots, v_k\}$ is chosen for a generic granule $j$. In case of aggregate coercion functions, an aggregate function, such as the average or the sum, is applied to the values $\{v_1, \ldots, v_k\}$ to compute the value of granule $j$.[5] In case of user-defined coercion function the method to convert from one granularity to the other is completely left to the user.

[5]Obviously these functions apply in case of set of values for which such functions are defined, such as, for example integers.

**Example 5** Let $\mathtt{t}_2 = temporal_{months}(\mathtt{integer})$ such that $\{\langle I_{2/1998}, 4\rangle, \langle I_{4/1998}, 5\rangle, \langle I_{9/1998}, 3\rangle\} \in T\_Eval(\mathtt{t}_2)$ where $months(2/1998) = I_{2/1998}$ and so on. Moreover, let $\mathtt{t}_1 = temporal_{years}(\mathtt{integer})$. An example of aggregate coercion function which maps values of type $\mathtt{t}_2$ into values of type $\mathtt{t}_1$ is $\{\langle I_{1998}, 4\rangle\} = \mathtt{avg}(\{\langle I_{2/1998}, 4\rangle, \langle I_{4/1998}, 5\rangle, \langle I_{9/1998}, 3\rangle\})$. Moreover an example of selective coercion function which maps values of type $\mathtt{t}_2$ into values of type $\mathtt{t}_1$ is $\{\langle I_{1998}, 3\rangle\} = \mathtt{last}(\{\langle I_{2/1998}, 4\rangle, \langle I_{4/1998}, 5\rangle, \langle I_{9/1998}, 3\rangle\})$. ◇

In order to ensure substitutability some restrictions are imposed on the ISA relationship, formalized by the following rule.

**Rule 1** (*Attribute Refinement*). Let $c_1$ and $c_2$ be two classes such that $c_2 <_{ISA} c_1$. Moreover, let $a$ be an attribute of class $c_1$ whose domain is of type $\mathtt{t}_1$) in $c_1$. Then, $a$ can be refined in $c_2$ by an attribute of type $\mathtt{t}_2$) iff one of the following conditions holds.

1. $\mathtt{t}_2 \leq_T \mathtt{t}_1$;
2. $\mathtt{t}_1 = temporal_G(\mathtt{t}'_1)$, $\mathtt{t}_2 = temporal_H(\mathtt{t}'_2)$, $\mathtt{t}'_2 \leq_T \mathtt{t}'_1$, $H \preceq G$ and an appropriate coercion function $C$ is associated to attribute $a$ in $c_2$. △

The following proposition proves that Rule 1 ensures substitutability.

**Proposition 2** *Given an ISA relationship, if Rule 1 holds, then type substitutability is ensured.* △

**Proof Sketch** If condition 1 holds, $\mathtt{t}_2$ is a subtype of $\mathtt{t}_1$. Then, because of Proposition 1, the values of attribute $a$ of instances of $c_2$ are legal values for attribute $a$ of instances of $c_1$, and thus substitutability is ensured. If condition 2 holds the coercion function ensures substitutability of instances of type $c_2$ with respect to instances of type $c_1$. ○

## 5. An Illustrative Example

In this section an example of an inheritance hierarchy including several temporal granularities is presented.

**Example 6** Let `Course` be a class storing the information concerning university courses. For each course we are interested in representing the `name`, the professor (`T_professor`), the teaching assistant (`T_assistant`) and the students taking the course (`P_students`). Information are stored with different granularities. For example, the professor teaching the course is the same during one year, whereas the teaching assistant is the same during a semester. Moreover, for internal statistics, for each course the set

$\langle$attr_spec$\rangle$ ::= $\langle$attribute_name:domain_type$\rangle$ | $\langle$ref attribute_name:domain_type [$\langle$ coerc_func $\rangle$]$\rangle$

$\langle$coerc_func$\rangle$ ::= $\langle$selective_coerc_func$\rangle$ | $\langle$aggregate_coerc_func$\rangle$ | $\langle$user-def_coerc_func$\rangle$

$\langle$selective_coerc_func$\rangle$ ::= first | last | Proj(index)

$\langle$aggregate_coerc_func$\rangle$ ::= min | max | avg | sum

$\langle$user-def_coerc_func$\rangle$ ::= meth_inv

**Figure 1. BNF grammar**

of students attending the course during one year is stored. In the following let Professor, Researcher, Student and Person be classes and let $days$, $weeks$, $months$, $semesters$ and $years$ be granularities such that $days \preceq months \preceq semesters \preceq years$ and $days \preceq weeks$.

```
class Course
  {
  name:string;
  T_professor:$temporal_{years}$(Professor);
  T_assistant:$temporal_{semesters}$(Researcher);
  P_students:$temporal_{years}$(set<Student>);
  }
```

Courses are partitioned into three types: introductory courses (I_Course), experimental courses (E_Course) and theoretical courses (T_Course). Each class representing each of the course type is a subclass of the class Course, that is, I_Course $\leq_{ISA}$ Course, E_Course $\leq_{ISA}$ Course and T_Course $\leq_{ISA}$ Course. Each class inherits all the attributes of the Course class and redefines some of them according to Rule 1. Moreover they introduce proper attributes. In the following only redefined or proper attributes are reported.

```
class I_Course:  Course
  {
  T_assistant:$temporal_{years}$(Researcher);
  P_students:($temporal_{days}$(set<Student>),
              count_students());
  room:  $temporal_{years}$(integer);
  lab:  $temporal_{weeks}$(string);
  }

class E_Course:  Course
  {
  T_assistant:($temporal_{months}$(Researcher),
               present());
  expert:$temporal_{days}$(Person);
  lab:$temporal_{months}$(string);
  }

class T_Course:  Course
  {
  T_assistant:($temporal_{months}$(Researcher),
               present());
  room:  $temporal_{months}$(integer);
  }
```

Let us discuss each class separately. As far as introductory courses are concerned, the teaching assistant is the same for the whole year, thus I_Course redefines the T_assistant attribute whose type is now $temporal_{years}$(Researcher). Type substitutability is obviously ensured since for the year the teaching assistant is the same for each semester. Moreover, for internal statistics the students participating to each daily class must be stored. Since $days$ is finer than $years$ a coercion function must be defined for the refined attribute in class I_Course, here the user-defined coercion function count_students() is defined. A possible implementation for count_students() could be a method computing the percentage of days in which the student has participated to the course, for each student in the union of the value of P_students over the days of the year. If this percentage is greater than fifty percent, the student is returned in the set given as result. Similarly, both E_Course and T_Course classes redefine the T_assistant attribute with a granularity finer than the one in the superclass. Thus an appropriate coercion function is associated with the attribute. A possible implementation for the present() user-defined coercion function could be a method which computes for each semester the researcher who did most of lectures for the course. Finally, each class introduces some proper attributes. For introductory courses both a room and a laboratory (lab) are needed. The room is the same for the whole year, while the laboratory can change each week. Experimental courses only have a laboratory and it is the same for each month. Moreover an expert can give daily seminars to the students. Theoretical courses need a room for lectures and the same room is assigned to a theoretical course in a month. $\diamond$

## 6. Conclusions

In handling temporal information, the need often arises of managing temporal data expressed according to different granularities. Those data should in many cases be compared and used by the same application program. A temporal data model should thus offer the possibility of handling multiple temporal granularities. In this paper, we have formally defined a temporal object-oriented data model supporting multiple granularities, and we have revisited on a formal basis issues related to inheritance, type refinement and substitutability.

We are extending the work presented in this paper along several directions. First, all operations on objects should be re-examined in this new context: for example, different notions of object equality can be devised when comparing two objects whose states have as components temporal values expressed at different granularities. Moreover, we are studying the problem of querying this kind of objects and developing a query language for the proposed data model. Finally, we are investigating how the proposed model can be effectively implemented on top of an ODMG-compliant object-oriented DBMS. We are indeed extending the temporal extension of the ODMG data model we have proposed [2] to handle multiple granularities.

## References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] E. Bertino, E. Ferrari, G. Guerrini, and I. Merlo. Extending the ODMG Object Model with Time. In E. Jul, editor, *Proc. Twelfth European Conference on Object-Oriented Programming*, number 1445 in Lecture Notes in Computer Science, pages 41–66, 1998.

[3] C. Bettini, C. Dyreson, W. Evans, and R. Snodgrass. A Glossary of Time Granularity Concepts. In *Temporal Databases: Research and Practice*, number 1399 in Lecture Notes in Computer Science, pages 406–413, 1998.

[4] C. Bettini, X. S. Wang, and S. Jajodia. A General Framework and Reasoning Models for Time Granularity. In *Proc. of the Third International Workshop on Temporal Representation and Reasoning (TIME'96)*, 1996.

[5] C. Bettini, X. S. Wang, and S. Jajodia. Satisfiability of Quantitative Temporal Constraints with Multiple Granularities. In *Proc. of 3rd International Conference on Principle and Practice of Constraint Programming*, 1997.

[6] L. Cardelli. Types for Data Oriented Languages. In J. W. Schmidt, S. Ceri, and M. Missikoff, editors, *Proc. First Int'l Conf. on Extending Database Technology*, Lecture Notes in Computer Science, pages 1–15, 1988.

[7] R. Cattel, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade. *The Object Database Standard: ODMG 2.0*. Morgan-Kaufmann, 1997.

[8] C. Combi, G. Cucchi, and F. Pinciroli. Applying Object-Oriented Technologies in Modeling and Querying Temporally Oriented Clinical Databases Dealing with Temporal Granularity and Indeterminancy. *IEE Transactions on Inforamtion Technology in Biomedicine*, 1(2):100–127, 1997.

[9] M. T. Ozsu, R. Peters, D. Szafron, B. Irani, A. Lipka, and A. Munoz. TIGUKAT: A Uniform Behavioral Objectbase Management System. *VLDB Journal*, 4(3):445–492, 1995.

[10] E. Rose and A. Segev. TOODM - A Temporal Object-Oriented Data Model with Temporal Constraints. In *Proc. Tenth Int'l Conf. on the Entity-Relationship Approach*, pages 205–229, 1991.

[11] R. T. Snodgrass. *The TSQL2 Temporal Query Language*. Kluwer Academic Publisher, 1995.

[12] X. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical Design for Temporal Databases with Multiple Granularities. *ACM Transactions on Database Systems*, 22(2):115–170, 1997.

[13] Y. Wu, S. Jajodia, and X. S. Wang. Temporal Database Bibliography Update. In *Temporal Databases: Research and Practice*, number 1399 in Lecture Notes in Computer Science, pages 338–366, 1998.

[14] G. Wuu and U. Dayal. A Uniform Model for Temporal and Versioned Object-Oriented Databases. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases: Theory, Design, and Implementation*, pages 230–247. Benjamin/Cummings, 1993.