Querying Multiple Temporal Granularity Data

Isabella Merlo[1] Elisa Bertino[2] Elena Ferrari[2] Shashi Gadia[3] 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

[3]Department of Computer Science
Iowa State University - Iowa
gadia@cs.iastate.edu

Abstract

Managing and querying information with varying temporal granularities is an important problem in databases. Although there is a substantial body of work on temporal granularities for the relational data model [11], a comprehensive framework is lacking for the object-oriented paradigm. To the best of our knowledge, a formal treatment of temporal queries with multiple granularities has not been considered in the literature. In this paper we make a step in this direction. We formally introduce the syntax and semantics of expressions involving data with multiple granularities, comparison between data with different granularities, and conversion of data from one granularity to another. We believe that this is an important step towards the development of an object-oriented query language that supports multiple granularities.

Keywords: *temporal granularities, temporal object databases, temporal object query languages.*

1. Introduction

Managing and relating temporal information at different time units is an important issue in many applications and research areas, among them temporal databases. The introduction of different *temporal granularities* [3], to store the historical information in a temporal database has a twofold advantage. The use of the appropriate granularities to represent data can save storage and allows one to establish relevant integrity constraints so that data cannot be changed more often than the specified granularity.

Some interesting issues arise in extending a database model to store and query data with multiple granularities. The introduction of temporal granularities in both the relational and the object-oriented database model, has been investigated in previous works [8, 9, 10, 11, 12]. TSQL-2 [11], a standard for the relational model, provides limited facilities for querying temporal data with varying granularities. To the best of our knowledge, no extensive work has been done to formally define how temporal data with different granularities can be queried in the object-oriented context. The proposals for temporal object-oriented query languages [5, 6] do not support facilities for querying data with different granularities. In particular a consolidated standard definition does not exist for the object-oriented data model, even though a preliminary definition can be found in our previous work [8]. Recently, we have developed an extension of [8] which has been submitted for publication [1].

In this paper we make a step towards the formal treatment of granularities for object-oriented databases. We define the syntax and the semantics of expressions involving data with multiple temporal granularities. The temporal model we refer to throughout the paper is an extension of an ODMG-compliant object-oriented model [4]. This model has been presented in our previous work [8]. The expressions we introduce in this paper are an extension of OQL [4] binary expressions. They can be regarded as a step forward the definition of an object-oriented query language for querying multiple temporal granularity data. The semantics of comparisons and conversions of data with different granularities is formally defined in the paper. In order to answer complex queries, the expressions include several types of comparison operators. For instance, it is possible to answer if a given inequality/equality *always* or *sometimes* holds, as well as to determine *when* it holds. Finally, several different types of conversions for data from one granularity to another are considered.

The remainder of this paper is organized as follows. Section 2 introduces some preliminary concepts. In Section 3 the syntax of expressions is presented. The corresponding semantics is formally defined in Section 4. Section 5 presents an advanced form of expressions, in which conversions among granularities are associated with operators, in order to increase the expressive power of the expressions. Section 6 shows how the defined expressions can be used in OQL queries. Finally, Section 7 concludes the paper.

2. Preliminaries

In this section we introduce, as a background, some preliminary concepts on temporal granularities and on the set of types and values of the expressions of our language. Those notions are mainly taken from two previous papers [3, 8].

2.1. Multiple Granularities

Issues concerning temporal granularities are a recent research topic in the temporal database area. One of the best known work dealing with temporal granularities is the glossary of time granularity concepts [3]. In the glossary all the main concepts concerning temporal granularities are formally defined without referring to any particular data model. To be as general as possible we refer to those definitions.

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. In our context the time domain is the pair ($\mathbb{N}$,$\leq$), where $\mathbb{N}$ is the set of natural numbers and represents the set of *time instants*, and $\leq$ is the order on $\mathbb{N}$. Temporal granularities are formally defined as follows.

Definition 1 (*Granularity*)[3]. Let $\mathcal{IS}$ be an index set, that is, a set of positive 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* $G(i)$ is identified by an integer. The first condition in Definition 1 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. Figure 1 graphically illustrates those concepts.

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 $mm/dd/yyyy$, $months$ are in the form $mm/yyyy$ and so on.

A *finer than* relationship can be defined among temporal granularities, with the following meaning.

Definition 2 (*Finer Than Relationship*)[3]. 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 define comparison operators between values at different granularities. Note that the finer than relationship establishes a partial order on a set of granularities $\mathcal{G}$.

Finally, we introduce the notion of *temporal element* with respect to a granularity. The notion of temporal element is formally introduced in [7]. A temporal element is a finite union of intervals. Intuitively, every subset of the set of granules associated with a granularity G is a temporal element. Thus, in what follows, given a granularity G, a temporal element T_G with respect to such granularity is defined as $T_G = \{G(i) \mid i \in I, I \subseteq \mathcal{IS}\}$.

2.2. Temporal Types and Values

In this section, temporal types related to different granularities and the notion of set of values of a given type are formally defined according to the set of temporal types. In what follows we do not discuss the set of object types, set of literal types, and so on, of our type system, because they are not relevant in the present context. We refer the interested reader to [8, 1] for a detailed description of the reference model.

In [8, 1] object types can be defined through classes and temporal information is associated with attributes. We consider a classical notion of class [2] where, in order to store temporal information, the type of an attribute can be $temporal_G(\texttt{t})$. Thus, in what follows the terms class and object type are used as synonyms.

We assume that a set $\mathcal{ST}$ of types is given. Such set includes object and literal types.[1] For each type $\texttt{t} \in \mathcal{ST}$ and granularity $G \in \mathcal{G}$, a corresponding temporal type, $temporal_G(\texttt{t})$, is defined. Intuitively, instances of type $temporal_G(\texttt{t})$ are partial functions from granules of G to instances of type $\texttt{t}$.

[1] Example of types belonging to $\mathcal{ST}$ are `integer`, `person`, and so on.

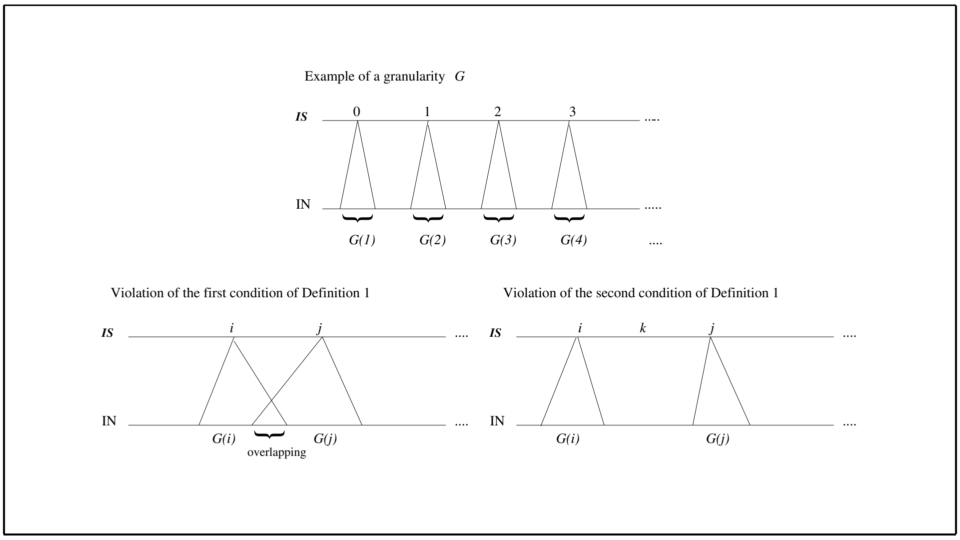

Figure 1. Graphical representation of granularities

Definition 3 (*Temporal Types*)[8]. Let $\mathtt{t} \in \mathcal{ST}$ be a type and $G \in \mathcal{G}$ be a temporal granularity, then $temporal_G(\mathtt{t})$ is the temporal type corresponding to type t and granularity G. □

Note that, according to the previous definition, temporal types cannot be nested. That is, if $\mathtt{t} \in \mathcal{ST}$ is a type and G and $H \in \mathcal{G}$ are two granularities, $temporal_G(temporal_H(\mathtt{t}))$ is not a legal type of our model.

Example 1 Let `integer` and `person` be two types in $\mathcal{ST}$, and let $years$ and $months$ be two granularities in $\mathcal{G}$, then examples of temporal types are $temporal_{years}(\mathtt{integer})$ and $temporal_{months}(\mathtt{person})$.◇

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 the following, the whole set of the types provided by our model is denoted as $\mathcal{T}$. $\mathcal{T}$ is an extension of the set of given types $\mathcal{ST}$, with temporal types. Each type $\mathtt{t} \in \mathcal{ST}$ still belongs to $\mathcal{T}$, that is, $\mathcal{ST} \subseteq \mathcal{T}$. $\mathcal{T}$ is partitioned into three disjoint sets, that is, $\mathcal{T} = \mathcal{TT} \cup \mathcal{OTT} \cup \mathcal{LTT}$, where $\mathcal{TT}$ is the set of temporal types, $\mathcal{OTT}$ is the set of object types, and $\mathcal{LTT}$ is the set of literal types.

In what follows, 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 t and a time instant t, returns the set of identifiers of objects belonging to type t at time t. In what follows, given a literal type t, $dom(\mathtt{t})$ denotes the set of values of that type. 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 4 (*Legal Values of a Non-Temporal Type with Respect to the Time Domain*)[8]. 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 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 value 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 5 (*Legal Values of a Non-Temporal Type with Respect to a Granule*)[8]. 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 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. $\square$

Example 2 Let `department` be an object type such that $Eval_{years}(\texttt{department},\ 1997)^2 = \{\texttt{d}_1, \texttt{d}_2,\ \texttt{d}_3\}$. Then, for each instant belonging to 1997, $\texttt{d}_1$,$\texttt{d}_2$, and $\texttt{d}_3$ must exist. Formally, $Eval_{years}(\texttt{department},\ 1997) = \bigcap_{t \in I} Eval(\texttt{department},\ t)$, 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 the object type `department` of that year. $\diamond$

We are now ready to define the set of legal values for temporal types with respect to a given time instant $t \in \mathbb{N}$. Intuitively, a legal value of a temporal type $temporal_G(\texttt{t})$ is a partial function from granules of G to legal values of type `t`.

Definition 6 (*Temporal Type Legal Values*)[8]. Let $temporal_G(\texttt{t}) \in \mathcal{TT}$ be a temporal type and $t \in \mathbb{N}$ be a time instant, then $T\_Eval(temporal_G(\texttt{t}), t)$ denotes the set of legal values of type $temporal_G(\texttt{t})$ at instant t:

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

$\square$

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

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

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 5) to temporal types as follows:

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

The following example clarifies the above definitions.

Example 3 Let `department` be an object type such that: $D = \bigcup_{i \in \mathcal{IS}} Eval_{years}(\texttt{department},\ i) = \{\texttt{d}_1, \texttt{d}_2,\ \texttt{d}_3,\ \ldots,\ \texttt{d}_n\}$. Then, according to Definition 6:

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

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

Examples of functions, denoted as sets of pairs, in $T\_Eval(temporal_{years}(\texttt{department}),\ t)$ are: $\{\langle 1992,\ \texttt{d}_1\rangle, \langle 1993,\ \texttt{d}_4\rangle\}$ and $\{\langle 1995,\ \texttt{d}_2\rangle, \langle 1998,\ \texttt{d}_2\rangle\}$. In those functions 1992, 1993, and so on, are the labels in the form $yyyy$, denoting years. $\diamond$

3. Expressions

In this section we present the formal syntax of expressions that can be built in our language. Such expressions can be used in querying data, similarly to expressions used in OQL queries.

We restrict ourselves to the expressions that can be built starting from temporal values involving integers and booleans. Such expressions are simple, but they allow one to focus on temporal aspects. The extension to general types of expressions, involving for example characters, can be easily done. Thus, we consider temporal values whose inner type is `integer` or `boolean`, that is, values belonging to the temporal types $temporal_G(\texttt{integer})$ and $temporal_G(\texttt{boolean})$, $G \in \mathcal{G}$. The BNF grammar of temporal expressions is given in Figure 2. Such expressions are a generalization of OQL binary expressions involving integers and booleans [4].

Terminal symbols in Figure 2 have the following meaning. Symbol `var` denotes an object-denoting variable. Symbol `class_name` denotes a class name. Symbols `int_value` and `bool_value` denote an element belonging to $dom(\texttt{integer})$ and $dom(\texttt{boolean})$, respectively. For example, 1 or 138 for `int_value` and *true* or *false* for `bool_value`. Symbols `int_temp_value` and `bool_temp_value` denote a temporal value whose inner type is `integer` and `boolean`, respectively. That is, `int_temp_value` represents functions belonging to the following set:

$$
\begin{aligned}
&T\_Eval(temporal_G(\texttt{integer})) = \\
&\quad \{f \mid f = \bar{f} \circ G \text{ such that } \bar{f} : 2^{\mathbb{N}} \rightarrow dom(\texttt{integer}) \\
&\quad \text{is a partial function such that for each } i \in \mathcal{IS} \\
&\quad \text{if } \bar{f}(G(i)) \text{ is defined then } \bar{f}(G(i)) \in dom(\texttt{integer})\}.
\end{aligned}
$$

Similarly, `bool_temp_value` represents functions belonging to the set $T\_Eval(temporal_G(\texttt{boolean}))$. Finally, square brackets denote optional symbols, whereas parenthesis denote the arbitrary repetition of what is in between.

Let v be a temporal value. Since v is a partial function, there may exist $i \in \mathcal{IS}$ such that $v(G(i)) = \perp$. We consider such *unknown* information as *null* values in OQL. Thus, they are considered as zero in integer expressions and three-values logics is adopted for boolean expressions.

In Figure 2 the syntax of path expressions denoting object navigation is defined. We consider path expressions where attribute domains are correct with respect to type checking. That is, let o be an object of class c and let $o.a_1.a_2$

⟨exp⟩ ::= ⟨int_exp⟩ | ⟨bool_exp⟩ | ⟨temp_exp⟩

⟨int_exp⟩ ::= int_value | int_temp_value | path_exp | ⟨int_exp⟩ ⟨op_int⟩ ⟨int_exp⟩

⟨path_exp⟩ ::= var { .attribute_name }

⟨op_int⟩ ::= + | - | * | / | mod

⟨bool_exp⟩ ::= bool_value | bool_temp_value | path_exp | ⟨int_exp⟩ ⟨op_bool_int⟩ ⟨int_exp⟩ | ⟨bool_exp⟩ ⟨op_bool⟩ ⟨bool_exp⟩

⟨op_bool_int⟩ ::= $=|!=|<|<=|>|>=|=_A|!=_A|<_A|<=_A|>_A|>=_A|=_S|!=_S|<_S|<=_S|>_S|>=_S$

⟨op_bool⟩ ::= and | or

⟨temp_exp⟩ ::= ⟨int_exp⟩ ⟨op_temp_int⟩ ⟨int_exp⟩

⟨op_temp_int⟩ ::= $=_T|!=_T|<_T|<=_T|>_T|>=_T$

Figure 2. BNF grammar of expressions

be a path expression, than the attribute domain of a_1 in c is an object type, and the attribute domain of a_2 is an integer, boolean, or temporal type according to the expression in which $o.a_1.a_2$ appears. We do not discuss in detail the syntax and semantics of path expressions since it is out of the scope of this work. Besides path expressions, in Figure 2 three different types of expressions can be distinguished: integer, boolean, and temporal expressions. All of them can involve temporal values. Integer expressions and boolean expressions involving "classical" (that is, static) values have the usual semantics of classical integer and boolean expressions. In addition, we allow temporal values in integer and boolean expressions. The intuitive meaning of such expressions is the evaluation "point to point", that is, "instant to instant" in our context, of the expression. For each boolean operator *op* on integers we introduce three operators:

1. the *always* operator (op_A), whose intuitive meaning is to evaluate whether the relationship denoted by *op* always holds for the time period in which the two values are defined;

2. the *sometimes* operator (op_S), whose intuitive meaning is to evaluate whether the relationship denoted by *op* sometimes holds for the time period in which the two values are defined;

3. the *temporal* operator (op_T), whose intuitive meaning is to answer the following question: "when the relationship denoted by *op* holds?"

The first and the second kind of operators are a generalization of boolean expressions and the result of their evaluation is always *true* or *false*. By contrast, the third class of operators are the extension to a temporal context of boolean expressions and the result of their evaluation is a temporal element, thus, the expressions in which such operators appear are called *temporal expressions*. In what follows we present an example to clarify the previous concepts.

Example 4 Let `employee` be a class with a temporal attribute `salary` of type $temporal_{months}$(`integer`), and let $\mathtt{o}_1$ and $\mathtt{o}_2$ be two employee objects representing two married persons. Let $\mathtt{o}_1$`.salary=` $\{\langle 01/1999, 100\rangle, \langle 02/1999, 150\rangle, \langle 03/1999, 300\rangle\}$ and let $\mathtt{o}_2$`.salary=` $\{\langle 01/1999, 400\rangle, \langle 03/1999, 50\rangle\}$. If we are interested in knowing the family income, that is, the sum of the two salaries, we have to compute the value of the integer expression $\mathtt{o}_1$`.salary + `$\mathtt{o}_2$`.salary`. If we are interested in knowing which one of the two makes more money for each month, we have to compute $\mathtt{o}_1$`.salary` $>$ $\mathtt{o}_2$`.salary`. If we are interested in knowing if the husband, say $\mathtt{o}_1$, makes always more money than the wife, say $\mathtt{o}_2$, we should compute $\mathtt{o}_1$`.salary` $>_A$ $\mathtt{o}_2$`.salary`.[3] Finally, if we are interested in knowing when $\mathtt{o}_1$ made more money than $\mathtt{o}_2$ we have to compute $\mathtt{o}_1$`.salary` $>_T$ $\mathtt{o}_2$`.salary`. The resulting values of the presented expressions can be found in Example 5. $\diamond$

Some implications can be established among the different operators. For instance, if v_1 and v_2 are two temporal values and $v_1\,op_A\,v_2$ evaluates *true*, then $v_1\,op_S\,v_2$ evaluates *true*. In what follows we denote with $\mathcal{O}$ the set of classical comparison operators defined for integers, that is, $\mathcal{O} = \{=, !=, <, <=, >, >=\}$. Sets $\mathcal{O}_A$, $\mathcal{O}_S$, and $\mathcal{O}_T$ denote the *always*, *sometimes*, and *temporal* counterparts of the set $\mathcal{O}$. In the following section we will elaborate on the semantics of expressions formally defining which is the result of each expression of Example 4, even in the case in which the values are expressed with respect to different time granularities.

[3] We do not report the *sometimes* operator since it is similar to the always one.

$$\mathcal{E}_{int}[\![v_1 op v_2]\!] = \begin{cases} v_1 op^{int} v_2 & \text{if } v_1, v_2 \in dom(\texttt{integer}) \\ \{\langle G(i), v_1(G(i)) op^{int} v_2(G(i))\rangle \mid i \in \mathcal{IS}\} & \text{if } v_1, v_2 \in T\_Eval(temporal_G(\texttt{integer})) \\ \{\langle G(i), v_1(G(i)) op^{int} v_2\rangle \mid i \in \mathcal{IS}\} & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})) \text{ and } v_2 \in dom(\texttt{integer}) \\ \{\langle G(i), v_1 op^{int} v_2(G(i))\rangle \mid i \in \mathcal{IS}\} & \text{if } v_1 \in dom(\texttt{integer}) \text{ and } v_2 \in T\_Eval(temporal_G(\texttt{integer})) \\ \mathcal{E}_{int}[\![v_1 op R_G(v_2)]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})), \\ & \text{and } G \preceq H \\ \mathcal{E}_{int}[\![R_H(v_1) op v_2]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})), \\ & \text{and } H \preceq G \\ \mathcal{E}_{int}[\![R_K(v_1) op R_K(v_2)]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})), \\ & G \not\preceq H, H \not\preceq G, K \preceq G, \text{and } K \preceq H \\ \perp & \text{otherwise} \end{cases}$$

$$\mathcal{E}_{int}[\![e_1 op e_2]\!] = \mathcal{E}_{int}[\![e_1]\!]\, op^{int}\, \mathcal{E}_{int}[\![e_2]\!]$$

$$\mathcal{E}_{bool}[\![v_1 op_A v_2]\!] = \begin{cases} true & \text{if } \mathcal{E}_{bool}[\![v_1 op v_2]\!] = v, v \in T\_Eval(temporal_G(\texttt{boolean})), \text{and } \forall\, i\ s.t.\ v(G(i)) \neq \perp\ v(G(i)) = true \\ false & \text{if } \mathcal{E}_{bool}[\![v_1 op v_2]\!] = v, v \in T\_Eval(temporal_G(\texttt{boolean})), \text{and } \exists\, i\ s.t.\ v(G(i)) = false \\ \perp & otherwise \end{cases}$$

$$\mathcal{E}_{bool}[\![e_1 op_A e_2]\!] = \mathcal{E}_{bool}[\![\mathcal{E}_{int}[\![e_1]\!]\, op_A\, \mathcal{E}_{int}[\![e_2]\!]]\!]$$

$$\mathcal{E}_{bool}[\![v_1 op_S v_2]\!] = \begin{cases} true & \text{if } \mathcal{E}_{bool}[\![v_1 op v_2]\!] = v, v \in T\_Eval(temporal_G(\texttt{boolean})), \text{and } \exists\, i\ s.t.\ v(G(i)) \neq \perp\ v(G(i)) = true \\ false & \text{if } \mathcal{E}_{bool}[\![v_1 op v_2]\!] = v, v \in T\_Eval(temporal_G(\texttt{boolean})), \text{and } \forall\, i\ s.t.\ v(G(i)) = false \\ \perp & otherwise \end{cases}$$

$$\mathcal{E}_{bool}[\![e_1 op_S e_2]\!] = \mathcal{E}_{bool}[\![\mathcal{E}_{int}[\![e_1]\!]\, op_S\, \mathcal{E}_{int}[\![e_2]\!]]\!]$$

$$\mathcal{E}_{temp}[\![v_1 op_T v_2]\!] = \begin{cases} \{G(i) \mid v_1(G(i)) op^{bool} v_2(G(i)) = true\} & \text{if } v_1, v_2 \in T\_Eval(temporal_G(\texttt{integer})) \\ \{G(i) \mid v_1(G(i)) op^{bool} v_2 = true\} & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})) \text{ and } v_2 \in dom(\texttt{integer}) \\ \{G(i) \mid v_1 op^{bool} v_2(G(i)) = true\} & \text{if } v_1 \in dom(\texttt{integer}) \text{ and } v_2 \in T\_Eval(temporal_G(\texttt{integer})) \\ \mathcal{E}_{bool}[\![v_1 op_T R_G(v_2)]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})), \text{and } G \preceq H \\ \mathcal{E}_{bool}[\![R_H(v_1) op_T v_2]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})), \text{and } H \preceq G \\ \mathcal{E}_{bool}[\![R_K(v_1) op_T R_K(v_2)]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})), \\ & G \not\preceq H, H \not\preceq G, K \preceq G, \text{and } K \preceq H \\ \perp & \text{otherwise} \end{cases}$$

$$\mathcal{E}_{temp}[\![e_1 op_T e_2]\!] = \mathcal{E}_{bool}[\![\mathcal{E}_{int}[\![e_1]\!]\, op_T\, \mathcal{E}_{int}[\![e_2]\!]]\!]$$

Figure 3. Semantics of expressions

4. Semantics of Expressions

In what follows we focus on expressions involving temporal values. The semantics of the ones in which no temporal value appears is obvious and is the one supported by OQL.

Reasoning on temporal values we have to consider that they can be defined with respect to different granularities. One idea could be to force temporal values appearing in an expression to be expressed with respect to the same granularity. We believe that this is too restrictive. In addition, it is not coherent with the subtyping relationship among temporal types we have defined in [8]. In [8] the subtyping relationship establishes that a type $\mathtt{t}_2$ is a subtype of a type $\mathtt{t}_1$, $\mathtt{t}_1 = temporal_G(\mathtt{t}'_1)$, $\mathtt{t}_2 = temporal_H(\mathtt{t}'_2)$, if $\mathtt{t}'_2 \leq_{type} \mathtt{t}'_1$ and $G \preceq H$. Usually, in programming languages, values belonging to different types can appear in the same expression. In addition, temporal information expressed with respect to different granularities can be compared in several interesting ways, increasing the expressive power of our language. Thus, we believe that expressions involving temporal types related to different granularities have to be managed properly.

The semantic domain Exp is the set of well-formed expressions defined according to the BNF grammar of Figure 2. Such domain can be partitioned into three disjoint sets, that is, $Exp = Exp_{int} \cup Exp_{bool} \cup Exp_{temp}$, where Exp_{int} is the set of integer expressions, Exp_{bool} is the set of boolean expressions, and Exp_{temp} is the set of temporal expressions. In giving the semantics, we reason on each type of expression separately. We define three semantics functions, one for each kind of expressions.

- $\mathcal{E}_{int} : Exp_{int} \rightarrow T\_Eval(temporal_G(\mathtt{integer})) \cup dom(\mathtt{integer})$ gives the semantics of integer expressions;
- $\mathcal{E}_{bool} : Exp_{bool} \rightarrow T\_Eval(temporal_G(\mathtt{boolean})) \cup dom(\mathtt{boolean})$ gives the semantics of boolean expressions;
- $\mathcal{E}_{temp} : Exp_{temp} \rightarrow \{T_G \mid T_G$ is a temporal element$\}$ gives the semantics of temporal expressions.

A formal definition of such functions is reported in Figure 3. In Figure 3, v_1 and v_2 denote values. They represent integer, boolean or temporal values according to the expression that is under evaluation. Similarly, e_1 and e_2 denote more complex expressions. Symbol *op* denotes an integer or boolean operator depending on the semantic function which is being defined. In addition op_A, op_S, and op_T denote *always*, *sometimes* and *temporal* operators. The interpretation of an operator is denoted as op^{int} and op^{bool} for integer and boolean operators, respectively. R_G, R_H, and R_K denote different kinds of relaxing functions, which are formally defined in Definition 7.

Note that, in Figure 3, the definition of function $\mathcal{E}_{bool}$ for expressions of type v_1 *op* v_2 or e_1 *op* e_2 $op \in \mathcal{O}$ is not presented, because it is equal to the one for integer expressions, where the operation interpretation is op^{bool} and the type `integer` is substituted by `boolean`. We do not have specified the semantics of values that are expressions as well, since it is trivial. For instance, the semantic evaluation of an integer value is the value itself. Similarly, the semantics of path expressions is not discussed, since it is not relevant for the presented work.

The semantic function $\mathcal{E}_{int}$ for the standard operators belonging to $\mathcal{O}$ computes the evaluation of the expression "granule per granule", if the two values are expressed with respect to the same granularity. If the two values are expressed with respect to different granularities and the type of one value is subtype of the type of the other, function $\mathcal{E}_{int}$ "relaxes", through a *relaxing function R*, the value belonging to the most specific type to the type of the less specific value before the expression is evaluated. A *relaxing function*, given a temporal value of type $\mathtt{t}_2 = temporal_H(\mathtt{t}'_2)$ "relaxes" the value to a finer granularity G.

Definition 7 (*Relaxing 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, $\mathtt{t}'_2 \leq_{type} \mathtt{t}'_1$ and $G \preceq H$. A *relaxing function* R is a partial function defined as:

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

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

Given a temporal granularity H, for each granularity G such that $G \preceq H$, a relaxing function exists. In what follows, we denote each of those relaxing functions as R_G. Note that, in Definition 7, $\mathtt{t}_2$ is a subtype of $\mathtt{t}_1$, according to the subtype relationship we have defined in [8], thus the relaxing function can be seen as a *casting up* operation on the type of the given value.

In case the two values are expressed with respect to different granularities that are in some way comparable, that is, there exists a granularity K that is finer than both then two value granularities, the semantic functions "relax" the two values to such granularity K.[4] If none of the previous conditions is verified an undefined value is returned, that corresponds to an error detection.

Function $\mathcal{E}_{bool}$ on *always* and *sometimes* operators computes whether the considered inequality always or sometimes holds. Note that we consider the intersection of the domains of the functions representing the values, that is,

[4] If K exists, then it is the greatest lower bound of H and G in $\mathcal{G}$.

when both of them are defined, and we do not force the domain to be the same. This is in accordance with our intuition. If one asks: have John and his son always lived in the same house? The answer is yes if they have lived in the same house after John's son was born. In this case the domain of the temporal function corresponding to the value storing the history of the address of John's son is strictly included in the domain of the temporal function corresponding to the value storing the history of the address of John. Finally, function $\mathcal{E}_{temp}$ computes the temporal element in which the considered inequality holds.

In the following example we give the semantics of the temporal expressions of Example 4. In addition the semantics of some expressions involving values related to different granularities is given.

Example 5 According to the semantic functions defined in Figure 3, the semantics of the expressions of Example 4 is the following. We recall that $\texttt{o}_1\texttt{.salary=}\{\langle 01/1999, 100\rangle, \langle 02/1999, 150\rangle, \langle 03/1999, 300\rangle\}$ and $\texttt{o}_2\texttt{.salary=}\{\langle 01/1999, 400\rangle, \langle 03/1999, 50\rangle\}$.

$$\begin{aligned}
&\mathcal{E}_{int}\,[\![\,\texttt{o}_1\texttt{.salary} + \texttt{o}_2\texttt{.salary}\,]\!] = \\
&\quad \{\langle 01/1999, 500\rangle, \langle 02/1999, 150\rangle, \langle 03/1999, 350\rangle\} \\
&\mathcal{E}_{bool}\,[\![\,\texttt{o}_1\texttt{.salary} > \texttt{o}_2\texttt{.salary}\,]\!] = \\
&\quad \{\langle 01/1999, false\rangle, \langle 03/1999, true\rangle\} \\
&\mathcal{E}_{bool}\,[\![\,\texttt{o}_1\texttt{.salary} >_A \texttt{o}_2\texttt{.salary}\,]\!] = false \\
&\mathcal{E}_{bool}\,[\![\,\texttt{o}_1\texttt{.salary} >_S \texttt{o}_2\texttt{.salary}\,]\!] = true \\
&\mathcal{E}_{temp}\,[\![\,\texttt{o}_1\texttt{.salary} >_T \texttt{o}_2\texttt{.salary}\,]\!] = \{03/1999\}
\end{aligned}$$

Consider now two classes `region` and `city`. Suppose that class `region` has a temporal attribute `temperature` of type $temporal_{months}(\texttt{integer})$ storing the value of the temperature[5] in that region for each month, according to some formula.[6] Moreover, suppose that class `city` has a temporal attribute `temperature` of type $temporal_{days}(\texttt{integer})$ storing the value of the temperature in a city for each day.

Let `r` and `c` be two objects belonging to `region` and `city`, respectively, such that $\texttt{r.temperature} = \{\langle 12/1999, 5\rangle, \langle 01/2000, 7\rangle, ...\}$ and $\texttt{c.temperature} = \{\langle 01/12/1999, 5\rangle, \langle 02/12/1999, 10\rangle, \langle 03/12/1999, 8\rangle, \langle 04/12/1999, 1\rangle, ...\}$.[7] Suppose that `c` is a city in region `r` and that we are interested in knowing when such city has had a temperature greater than the value of its region, we should compute $\texttt{r.temperature} <_T \texttt{c.temperature}$, whose semantics is

$$\begin{aligned}
&\mathcal{E}_{temp}\,[\![\,\texttt{r.temperature} <_T \texttt{c.temperature}\,]\!] = \\
&\mathcal{E}_{temp}\,[\![\,R_{days}(\texttt{r.temperature}) <_T \texttt{c.temperature}\,]\!] = \\
&\{days(i) \mid \texttt{r.temperature}(days(i)) <^{bool} \\
&\texttt{c.temperature}(days(i))\} = \\
&\{02/12/1999, 03/12/1999, ...\}
\end{aligned}$$

where $R_{days}(\texttt{r.temperature}) = \{\langle 01/12/1999, 5\rangle, \langle 02/12/1999, 5\rangle, \langle 03/12/1999, 5\rangle, \langle 04/12/1999, 5\rangle, ..., \langle 01/01/2000, 7\rangle, \langle 02/01/2000, 7\rangle, ...\}$.

◇

In the previous example, of course, one can argue that if the temperature of a region has been stored with respect to months, it is not realistic to deduce that the temperature in such region was the same for every day of the month. This problem is strictly correlated with temporal granularity issues. In fact, storing the value of an attribute with respect to a granularity is somehow deciding the temporal precision associated with the information. If the attribute granularity is $months$, then for each granularity finer than $months$, such as, for instance, $days$ this information is imprecise. However, since a value has been associated with each month, this information is the only one we have "closer" to the value of each day. Consider, for example, the value of the temperature of a region stored monthly, probably such value is an average value, thus, it is realistic to deduce that every day the value of the temperature has been very close to the stored value. This implies that "relaxing" the value to the granularity $days$ is a reasonable choice and allows one to compare this value with the ones of the cities.

5. Advanced Expressions

In order to convert values from a given granularity into values of a coarser granularity in a meaningful way, in [8] we have introduced *coercion functions*, whose formal definition is the following.

Definition 8 (*Coercion Function*)[8]. Let $\texttt{t}_1 = temporal_G(\texttt{t}'_1)$ and $\texttt{t}_2 = temporal_H(\texttt{t}'_2)$ be two temporal types such that, $\texttt{t}'_2 \leq_{type} \texttt{t}'_1$[8] and $H \preceq G$. A coercion function C is a partial function defined as:

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

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

In [8], coercion functions were associated with an attribute definition allowing one to specialize attribute domains in a type with a granularity finer than the one in the attribute domain to be redefined. In this paper such functions will be used to increase the expressive power of comparison operators.

[5] Temperatures are expressed in Celsius degrees.

[6] For example, that value is computed considering different temperatures at different time of the month in different established places.

[7] We do not report all the values for lack of space.

[8] $\leq_{type}$ denotes the subtyping relationship.

A large variety of coercion functions could be devised. We have developed a simple language for defining coercion functions. The syntax in BNF form of coercion functions is given in Figure 4. With reference to Figure 4 terminal symbol `index` denotes an element in $\mathcal{IS}$, and `meth_inv` denotes a method invocation. As specified in the BNF grammar of Figure 4, 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. The formal definition of coercion function classification can be found in [8]. In Figure 4 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.

Let $\{i_1, \ldots, i_k\}$ be the set of indexes such that $H(i_k) \subseteq G(j)$ and let v be a temporal value of type $temporal_H(\mathtt{t})$ such that $v(H(i_k)) = v_k$. Then, 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.[9] In case of user-defined coercion functions, the method to convert from one granularity to the other is completely specified by the user.

Example 6 Let $\mathtt{t}_2 = temporal_{months}(\mathtt{integer})$ such that $\{\langle 02/1998, 4\rangle, \langle 04/1998, 5\rangle, \langle 09/1998, 3\rangle\} \in T\_Eval(\mathtt{t}_2)$. Moreover, let $\mathtt{t}_1 = temporal_{years}(\mathtt{integer})$. An example of the application of an aggregate coercion function which maps values of type $\mathtt{t}_2$ into values of type $\mathtt{t}_1$ is $\{\langle 1998, 1\rangle\} = \mathtt{avg}(\{\langle 02/1998, 4\rangle, \langle 04/1998, 5\rangle, \langle 09/1998, 3\rangle\})$. Moreover an example of the application of a selective coercion function which maps values of type $\mathtt{t}_2$ into values of type $\mathtt{t}_1$ is $\{\langle 1998, 3\rangle\} = \mathtt{last}(\{\langle 02/1998, 4\rangle, \langle 04/1998, 5\rangle, \langle 09/1998, 3\rangle\})$. $\diamond$

Note that actually, since we consider expressions involving boolean and integer values, the inner type in coercion functions can be `integer` or `boolean`. Thus, in what follows, given a coercion function C, and two granularities G and H such that $H \preceq G$, we denote with $C_{H\to G}$ the coercion function that coerces values with respect to granularity H into values of granularity G. For instance, the coercion function `avg` used in Example 6 is, more precisely, the $\mathtt{avg}_{months\to years}$ function.

Coercion functions add expressive power to our operators. We attach coercion function information to the operators in order to compute a more complex comparison operator. In addition to the coercion function, we optionally attach the granularity information to which this comparison has to refer. We call this new type of expressions *advanced expressions*. The BNF grammar and the semantics of such expressions can be found in Figure 4 and Figure 5, respectively. In the BNF grammar of Figure 5 the optional terminal symbol `gran` denotes an element in the set $\mathcal{G}$.

For each operator $op \in \mathcal{O} \cup \mathcal{O}_A \cup \mathcal{O}_S \cup \mathcal{O}_T$, we consider the advanced, op^C operator to which the coercion function is attached. When an operator op^C is applied to two values, first the coercion of the value expressed with respect to the finer granularity is computed, so that the two values are expressed with respect to the same granularity. Then the expression is evaluated according to the semantics defined in Figure 3.

The coercion function C can be complemented with a granularity K, denoted as $op^{C,K}$. In this case both the considered values have to be coerced to the granularity K which has to be coarser than the granularities of the two values. After that, the two values are expressed with respect to the same granularity, thus the expression can be evaluated according to the semantics defined in Figure 3. The following example clarifies those concepts.

Example 7 Consider again objects $\mathtt{o}_1$ and $\mathtt{o}_2$ of Example 4. If we are interested in knowing whether the average salary of the wife and the husband is always the same for every year we should compute $\mathtt{o}_1.\mathtt{salary} =_A^{\mathtt{avg},years} \mathtt{o}_2.\mathtt{salary}$. The semantic evaluation of that expression is:

$$\mathcal{E}_{bool}[\![\, \mathtt{o}_1.\mathtt{salary} =_A^{\mathtt{avg},years} \mathtt{o}_2.\mathtt{salary}]\!] = \mathcal{E}_{bool}[\![\, \mathtt{avg}_{months\to years}(\mathtt{o}_1.\mathtt{salary}) = \mathtt{avg}_{months\to years}(\mathtt{o}_2.\mathtt{salary})]\!] = false$$

since $\mathtt{avg}_{months\to years}(\mathtt{o}_1.\mathtt{salary}) = \{\langle 1999, 46\rangle\}$ and $\mathtt{avg}_{months\to years}(\mathtt{o}_2.\mathtt{salary}) = \{\langle 1999, 37\rangle\}$.

Consider again the two objects `r` and `t` of Example 5. If we are interested in knowing when the maximum temperature of region `r` is greater than the maximum temperature of city `c` for a whole month, we should compute the value of $\mathtt{r.temperature} <_T^{\mathtt{max}} \mathtt{c.temperature}$. Suppose that $\mathtt{max}_{days\to months}(\mathtt{c.temperature}) = \{\langle 12/1999, 10\rangle, \langle 01/2000, 11\rangle, \ldots\}$, then the semantic evaluation of the previous expression is the following:

$$\mathcal{E}_{temp}[\![\mathtt{r.temperature} <_T^{\mathtt{max}} \mathtt{c.temperature}]\!] = \mathcal{E}_{temp}[\![\mathtt{r.temperature} <_T \mathtt{max}_{days\to months}(\mathtt{c.temperature})]\!] = \{months(i) \mid \mathtt{r.temperature}(months(i)) <^{bool} \mathtt{c.temperature}(months(i))\} = \{12/1999, 01/2000, \ldots\}$$

We recall that $\mathtt{r.temperature} = \{\langle 12/1999, 5\rangle, \langle 01/2000, 7\rangle \ldots\}$. $\diamond$

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

⟨`coerc_func`⟩ ::= ⟨`selective_coerc_func`⟩ | ⟨`aggregate_coerc_func`⟩ | ⟨`user-def_coerc_func`⟩

⟨`selective_coerc_func`⟩ ::= `first` | `last` | `Proj(index)`

⟨`aggregate_coerc_func`⟩ ::= `min` | `max` | `avg` | `sum`

⟨`user-def_coerc_func`⟩ ::= `meth_inv`

⟨`ad_exp`⟩ ::= ⟨`ad_exp_bool`⟩ | ⟨`ad_exp_temp`⟩

⟨`ad_exp_bool`⟩ ::= ⟨`int_exp`⟩ $\langle\texttt{op\_bool\_int}\rangle^{\langle \texttt{coerc\_func}\rangle, [\texttt{gran}]}$ ⟨`int_exp`⟩

⟨`ad_exp_temp`⟩ ::= ⟨`int_exp`⟩ $\langle\texttt{op\_temp\_int}\rangle^{\langle \texttt{coerc\_func}\rangle, [\texttt{gran}]}$ ⟨`int_exp`⟩

Figure 4. BNF grammar of coercion functions and advanced expressions

$$\mathcal{E}_{bool}[\![v_1 op^C v_2]\!] = \begin{cases} \mathcal{E}_{bool}[\![C(v_1) op v_2]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})) \text{ and} \\ & G \preceq H \\ \mathcal{E}_{bool}[\![v_1 op C(v_2)]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})) \text{ and} \\ & H \preceq G \\ \perp & \text{otherwise} \end{cases}$$

$$\mathcal{E}_{bool}[\![e_1 op^C e_2]\!] = \mathcal{E}_{bool}[\![\mathcal{E}_{int}[\![e_1]\!] op^C \mathcal{E}_{int}[\![e_2]\!]]\!]$$

$$\mathcal{E}_{bool}[\![v_1 op^{C,K} v_2]\!] = \begin{cases} \mathcal{E}_{bool}[\![C_{G\to K}(v_1) op^{C,K} C_{G\to K}(v_2)]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})) \text{ and} \\ & G \preceq K \text{ and } H \preceq K \\ \perp & \text{otherwise} \end{cases}$$

$$\mathcal{E}_{bool}[\![e_1 op^{C,K} e_2]\!] = \mathcal{E}_{bool}[\![\mathcal{E}_{int}[\![e_1]\!] op^{C,K} \mathcal{E}_{int}[\![e_2]\!]]\!]$$

$$\mathcal{E}_{temp}[\![v_1 op_T^C v_2]\!] = \begin{cases} \mathcal{E}_{temp}[\![C(v_1) op_T v_2]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})) \text{ and} \\ & G \preceq H \\ \mathcal{E}_{temp}[\![v_1 op_T C(v_2)]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})) \text{ and} \\ & H \preceq G \\ \perp & \text{otherwise} \end{cases}$$

$$\mathcal{E}_{temp}[\![e_1 op_T^C e_2]\!] = \mathcal{E}_{temp}[\![\mathcal{E}_{temp}[\![e_1]\!] op_T^C \mathcal{E}_{int}[\![e_2]\!]]\!]$$

$$\mathcal{E}_{temp}[\![v_1 op_T^{C,K} v_2]\!] = \begin{cases} \mathcal{E}_{temp}[\![C_{G\to K}(v_1) op_T^{C,K} C_{H\to K}(v_2)]\!] & \text{if } v_1 \in T\_Eval(temporal_G(\texttt{integer})), \\ & v_2 \in T\_Eval(temporal_H(\texttt{integer})) \text{ and} \\ & G \preceq K \text{ and } H \preceq K \\ \perp & \text{otherwise} \end{cases}$$

$$\mathcal{E}_{temp}[\![e_1 op_T^{C,K} e_2]\!] = \mathcal{E}_{temp}[\![\mathcal{E}_{int}[\![e_1]\!] op_T^{C,K} \mathcal{E}_{int}[\![e_2]\!]]\!]$$

Figure 5. Semantics of advanced expressions

6. Illustrative Examples

In this section we present some examples of OQL queries using our expressions. Classes are defined in ODL syntax [4].

Example 8 Let `geo_item` be a class storing statistic information concerning geographical entities, which can be regions or cities. For each geographical item we are interested in representing the `name`, the `temperature`, and the `population`. The name does not change over time, whereas information related to the temperature and the population vary over time. Such information are stored according to different granularities. The temperature is stored monthly and the stored value is computed considering different temperatures at different times of the month in different established places. The population value is stored every year according to some average.

```
class geo_item (extent geo_items) {
  attribute string name;
  attribute temporal_{months}(integer) temperature;
  attribute temporal_{years}(integer) population;}
```

Geographical objects are specialized into two classes `region` and `city`. Each class represents a subclass of the class `geo_item`. In addition, each class inherits all the attributes of the class `geo_item` and defines some additional attributes.

```
class region extends geo_item (extent regions) {
  attribute set<city> tourist_cities;
  attribute set<city> art_cities;
  attribute set<city> fun_cities;}
```

```
class city extends geo_item (extent cities) {
  attribute region is_in;
  attribute temporal_{months}(integer) nbr_tourists;
  attribute temporal_{days}(integer) nbr_museums;
  attribute temporal_{weeks}(integer) nbr_expositions;
  attribute temporal_{days}(integer) nbr_concerts;
  attribute temporal_{days}(integer) nbr_discos;
  attribute temporal_{years}(integer) nbr_cinemas;
  attribute temporal_{days}(integer) nbr_theater_shows;
  attribute temporal_{days}(integer) nbr_markets;}
```

For each region, information related to the sets of tourist, art and fun cities are stored in the database. For what concerns cities, statistic tourist information are stored, such as, for instance, the number of open museums, the number of expositions, and so on, each one with a different granularity.

The following are OQL queries in which the operators we introduced are used.

- Return the ratio of the number of tourists with respect to the population of all the cities of region `Liguria`.

```
select struct(n:c.name,
              r:(c.nbr_tourists/c.population))
from cities c
where c.is_in.name = "Liguria"
```

The result of this query is a bag of structs where the first value is of type `string` and the second value is of type $temporal_{months}$`(integer)`. The temporal value is expressed in months, since the value of `c.population` is converted into months, then for each month the ratio is computed.

- Return the most touristic cities of each region.

```
select distinct struct(c:c.name,r:c.is_in.name)
from cities c
where forall t in
      (select k from cities k
      where c.is_in=k.is_in)
      c.nbr_tourists >=_A t.nbr_tourists
```

The result of the previous query returns a set of structures with two fields where both values are of type `string`. The first field represents those cities where the number of tourists is always, that is, for each month, greater than the number of tourists in all the other cities of the same region.

- Suppose we are interested in knowing which are the *fun* cities, where for fun cities we mean cities with an average number of museums which is always less than the average number of discos for each year, a maximum number of concerts which is always greater than or equal to the number of theater shows for every week, and a number of markets which is sometimes greater than the number of expositions. The following query finds out those cities:

```
select c
from cities c
where c.nbr_museums <_A^{avg,years} c.nbr_discos and
      c.nbr_concerts >=_A^{max,weeks} c.nbr_theater_shows
      and c.nbr_markets >_S c.nbr_expositions
```

◇

7. Conclusions

In this paper we have addressed the problem of representing and querying temporal data with multiple granularities. The issue of how to compare values with different granularities has been investigated and an extension of OQL-like expressions to facilitate such comparisons has been presented. The syntax of such expressions with the formal semantics have been defined. Finally, problems related to conversions of data from one granularity to another have been investigated and appropriate solutions to these problems have been presented.

This work is being extended in several directions. We are currently working on an extension of OQL path expressions to facilitate navigation among temporal objects whose attribute values are objects. We plan to provide an object query language to query temporal data with multiple granularities. Such a language will include temporal clauses using different granularities. The formal semantics of those clauses is currently under investigation. Finally, we will address issues related to the implementation of the extension we proposed on top of an object-oriented database supporting an OQL compliant query language.

References

[1] E. Bertino, E. Ferrari, G. Guerrini, and I. Merlo. An ODMG Compliant Temporal Object Model Supporting Multiple Granularity Management. Technical Report DISI-TR-00-08, Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, 2000. Submitted for publication.

[2] E. Bertino, G. Guerrini, and I. Merlo. Formalizing the ODMG Object Model. Technical Report DISI-TR-99-04, Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, 1999. Submitted for publication.

[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] 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.

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

[6] L. Fegaras and R. Elmasri. A Temporal Object Query Language. In *IEEE Proc. Fifth International Workshop on Temporal Representation and Reasoning*, IEEE Computer Society Press, pages 51–59, 1998.

[7] S. Gadia. A homogeneous relational model and query languages for temporal databases. *ACM Transactions on Database Systems*, 13(4):418–448, 1988.

[8] I. Merlo, E. Bertino, E. Ferrari, and G. Guerrini. A Temporal Object-Oriented Data Model with Multiple Granularities. In C. Dixon and M. Fischer, editors, *IEEE Proc. Sixth International Workshop on Temporal Representation and Reasoning*, IEEE Computer Society Press, pages 73–81, 1999.

[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] 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.