An Algebraic Representation of Calendars*
(Extended Abstract)

Peng Ning and **X. Sean Wang** and **Sushil Jajodia**
Department of Information and Software Engineering
George Mason University, Fairfax, Virginia, USA
{pning, xywang, jajodia}@gmu.edu

Abstract

This extended abstract uses an algebraic approach to define granularities and calendars. All the granularities in a calendar are expressed as algebraic expressions based on a single "bottom" granularity. The operations used in the algebra directly reflect the ways with which people construct new granularities from existing ones, and hence yield more natural and compact granularities definitions. The extended abstract also presents granule conversions between granularities in a calendar.

Introduction

System support for time has long been recognized to be important. Time is often represented in terms of closely related granularities (e.g., `year`, `month`, `day`) that are organized into calendars (e.g., Gregorian calendar). Reasoning and processing of time are usually performed on these representations. When the system allows users to define new granularities and calendars for the system to process, it is critical to have natural and flexible representation mechanisms. This extended abstract presents such a mechanism.

Natural representation is important not only for the ease of use. In many cases, it also allows more compact representations. As an example, consider the specification of leap years. A year is a leap one if the year (i.e., its number) is divisible by 4, but not divisible by 100 unless it's divisible by 400. A direct method of "coding" the leap year information is to have the above rule embedded in the definition of the granularity `year`. Unfortunately, it seems that all current proposals of granularity symbolic representations adopt an "explicit" method, namely list all the years in a 400 year period. Such a method is not scalable to granularities with large periods. In particular, the enumeration will take more storage and manipulating large periods may result in poor performance. In certain application systems such as a mobile computing environment, storage and processing time are both important.

In this extended abstract, we develop an algebraic representation for time granularities, which we call the *calendar algebra*. Each time granularity is defined as a mapping from its index set to the subsets of the time domain (BDE+98). We assume that there exists a "bottom" granularity known to the system. Calendar algebra operations are designed to generate new granularities from the bottom one or those already generated. The relationship between the operand(s) and the resulting granularities are encoded in the operations. The design of the operations aims at capturing the characteristics of calendars both naturally and expressively. For example, granularity `month` can be generated on the basis of granularity `day` by several calendar algebra operations. The first operation generates a granularity by partitioning all the days into 31-day groups, the second operation shrinks the second group of every 12 groups (which corresponds to February) by 3 days, the third step shrinks the fourth group of every 12 ones (which corresponds to April) by 1 day, etc.. To define `month` on the basis of `day` including all the leap year information, we only need nine operations (see the Calendar Algebra Operations section for details) without explicit enumeration of all the months in a period of 400 years (i.e., 4,800 months).

Calendars are then formalized on the basis of granularities defined by calendar algebra. The above mapping viewpoint of granularity represents granules using indexes, e.g., integers. However, people are used to relatively representations. For example, a particular day is represented in terms of the day in a `month`, and the month in a `year`. To formalize such representations, we develop label mappings.

The process of finding some granules in one granularity that has a particular relationship with a set of given granules in another granularity is called *granule conversion*. An example is to find all the business days in a given month. Granule conversion is essential to many applications like automatic evaluation of user queries, mixed granularities and multiple calendars support, and

*The work was partially supported by a grant from the U.S. Army Research Office under the contract number DAAG-55-98-1-0302 and a grant from the National Science Foundation with the grant number 9633541. The work of Wang was also partially supported by a Career Award from the National Science Foundation under the grant number 9875114.

rolling up along a time hierarchy in time series analysis or OLAP applications. We develop a generic method to solve the general granule conversion problem.

Preliminaries

We adopt some notions from (BDE+98).

Definition. (Time Domain) A *time domain* is a pair $(T, \leq)$ where T is a non-empty set of *time instants* and $\leq$ is a total order on T.

Definition. (Granularity) A *granularity* is a mapping G from a subset of integers, I (index set), to the subsets of the time domain such that for all i, j in I with $i < j$, if both $G(i)$ and $G(j)$ are non-empty, then (1) each element of $G(i)$ is less than all the elements of $G(j)$, and (2) for all k in I, $i < k < j$, $G(k)$ is non-empty.

Each non-empty set $G(i)$ is called a *granule* of granularity G.

To simplify the algebra, we use an extended notion of granularities. More specifically, a *labeled granularity* is a pair $(\mathcal{L}, G)$, where $\mathcal{L}$ is a subset of the integers, and G is a mapping from $\mathcal{L}$ to the subsets of the time domain such that for each pair of integers i and j in $\mathcal{L}$ with $i < j$, if $G(i) \neq \emptyset$ and $G(j) \neq \emptyset$, then (1) each element in $G(i)$ is less than every element of $G(j)$, and (2) for each integer k in $\mathcal{L}$ with $i < k < j$, $G(k) \neq \emptyset$. When $\mathcal{L}$ is exactly the integers, we call the granularity "full-integer labeled".

We will still use G to denote labeled granularities when no confusion arises.

Calendar Algebra Operations

In this section, we present a symbolic representation of granularities. The design of the representation scheme starts with the observation that granularities used in a calendar are not isolated, but rather are closely related. We thus design our symbolic representation based on some algebraic operations, called *calendar operations*, that capture these relationships. The symbolic representation is thus called the *calendar algebra*. Calendar operations generate new granularities by manipulating other granularities that are already generated. All granularities that are generated directly or indirectly from a single one (which will be the bottom granularity) form a calendar, and these granularities are related to each other through the operations that define them (from the bottom granularity). In practice, the choices for the bottom granularity include day, hour, second, microsecond and other granularities, depending on the accuracy required in each application context.

The calendar algebra consists of the following two kinds of operations: the *grouping-oriented operations* and the *granule-oriented operations*. The grouping-oriented operations group certain granules of a granularity together to form new granules in a new granularity, while the granule-oriented operations don't change the granules of a granularity, but rather make choices of which granules should remain in the new granularity.

Certain calendar operations will only work on full-integer labeled granularities, while others will be more easily defined and implemented using more flexible labeling.

To define the calendar algebra, a ***label-aligned sub-granularity*** relationship is needed. Formally, G_1 is a label-aligned sub-granularity of G_2 if the label set $\mathcal{L}_1$ of G_1 is a subset of the label set $\mathcal{L}_2$ of G_2 and for each i in $\mathcal{L}_1$ such that $G_1(i) \neq \emptyset$, we have $G_1(i) = G_2(i)$.

The grouping-oriented operations

The grouping operation Let G be a full-integer labeled granularity, and m a positive integer. The grouping operation $Group_m(G)$ generates a new granularity G' by partitioning the granules of G into m-granule groups and making each group a granule of the resulting granularity. For example, given granularity day, week can be generated by week = $Group_7$(day). Note here we assume that the day labeled 1 starts a week.

The altering-tick operation Let G_1, G_2 be full-integer labeled granularities, and l, k, m integers, where G_2 partitions[1] G_1, and $1 \leq l \leq m$. The altering-tick operation $Alter^m_{l,k}(G_2, G_1)$ generates a new granularity by periodically expanding or shrinking granules of G_1 in terms of granules of G_2. Since G_2 partitions G_1, each granule of G_1 consists of some contiguous granules of G_2. The granules of G_1 can be partitioned into m-granule groups such that $G_1(1)$ to $G_1(m)$ are in one group, $G_1(m+1)$ to $G_1(2m)$ are in the following group, and so on. The goal of the altering-tick operation is to modify the granules of G_1 so that the l^{th} granule of every aforementioned group will have $|k|$ additional (or fewer when $k < 0$) granules of G_2. For example, if G_1 represents 30-day groups (i.e., $G_1 = Group_{30}$(day)) and we want to add a day to every 12^{th} month (i.e., to make December to have 31 days), we may perform $Alter^{12}_{12,1}(\text{day}, G_1)$.

More specifically, for all $i = l + m \cdot n$, where n is an integer, $G_1(i)$ denotes the granule to be shrunk or expanded. The granules of G_1 are split into two parts at $G_1(0)$. When $i > 0$, $G_1(i)$ expands (or shrinks) by taking in (or pushing out) later granules of G_2, and the effect is propagated to later granules of G_1. On the contrary, when $i \leq 0$, $G_1(i)$ expands (or shrinks) by taking in (or pushing out) earlier granules of G_2, and the effect is propagated to earlier granules of G_1.

The altering-tick operation can be formally described as follows. For each integer i such that $G_1(i) \neq \emptyset$, let b_i and t_i be the integers such that $G_1(i) = \cup_{j=b_i}^{t_i} G_2(j)$. (The integers b_i and t_i exist because G_2 partitions G_1.) Then $G' = Alter^m_{l,k}(G_2, G_1)$ is the granularity such that for each integer i, let $G'(i) = \emptyset$ if $G_1(i) = \emptyset$, and other-

[1] G_2 partitions G_1 if each granule of G_1 is a union of some granules of G_2 and each granule of G_2 is a subset of a granule of G_1.

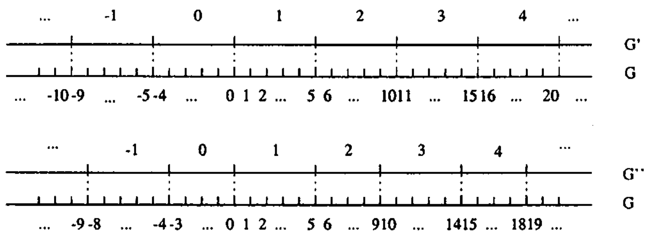

Figure 1: Grouping and altering tick operation

wise let

$$G'(i) = \bigcup_{j=b'_i}^{t'_i} G_2(j),$$

where

$$b'_i = \begin{cases} b_i + (h-1)\cdot k, & \text{if } i = (h-1)\cdot m + l, \\ b_i + h\cdot k, & \text{otherwise,} \end{cases}$$

$$t'_i = t_i + h \cdot k,$$

and

$$h = \lfloor \frac{i-l}{m} \rfloor + 1.$$

Fig. 1 shows an example of a grouping operation and an altering-tick operation. Granularity G' is defined by $G' = Group_5(G)$, while G'' defined by $G'' = Alter^2_{2,-1}(G, G')$, which means shrinking the second one of every two granules of G' by one granule of G.

An extension of the above operation is also used: When the parameter m is infinity (∞), the altering tick operation $Alter^k_{l,\infty}(G_2, G_1)$ means only altering the granule $G_1(l)$. For example, to add a leap second to the last minute of 1998, we may use $Alter^1_{x,\infty}(\texttt{second}, \texttt{minute})$, where x is the label of the last minute of 1998.

Shifting operation Let G be a full-integer labeled granularity, and m an integer. The shifting operation $Shift_m(G)$ generates a new granularity G' by shifting the labels of G by m positions. For each integer i, the granule $G'(i)$ will be the granule $G(i+m)$. The shifting operation can easily model time differences. Suppose granularity `GMT-hour` stands for the hours of Greenwich Mean Time. Then the hours of US Eastern Time can be generated from `GMT-hour` by

$$\texttt{USEast-Hour} = Shift_{-5}(\texttt{GMT-hour}).$$

Note. The grouping, altering-tick and shifting operations are collectively called *basic operations.* These basic operations are restricted to operate on full-integer labeled granularities (i.e., "regular" granularities), and the granularities generated by these operations are still full-integer labeled ones.

Combining operation Let G_1 and G_2 be granularities with label sets $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. The combining operation $Combine(G_1, G_2)$ generates a new granularity G' by combining all the granules of G_2 that are included in one granule of G_1 into one granule of G'. As an example, given granularities `b-day` and `month`, the granularity for business months can be generated by

$$\texttt{b-month} = Combine(\texttt{month}, \texttt{b-day}).$$

Anchored grouping operation Let G_1 and G_2 be granularities with label sets $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively, where G_2 is a label-aligned sub-granularity of G_1, and G_1 is a full-integer labeled granularity. The anchored grouping operation $Anchored\text{-}group(G_1, G_2)$ generates a new granularity G' by combining all the granules of G_1 that are between two granules of G_2 into one granule of G'. Granularity G_2 is called the ***anchor granularity*** of G_1 in this operation. The granules of G_2 divide the granules of G_1 into groups, and each group is made a resulting granule by the anchored grouping operation.

For example, each academic year at a certain university begins on the last Monday in August, and ends on the day before the beginning of the next academic year. Then, the granularity corresponding to the academic years can be generated by AcademicYear = *Anchored-group*(`day`, `lastMondayOfAugust`).

Granule-oriented operations

Subset operation The subset operation is designed to generate a new granularity by selecting an interval of granules from another granularity.

Let G be a granularity with label set $\mathcal{L}$, and m, n integers such that $m \leq n$. The subset operation $G' = Subset^n_m(G)$ generates a new granularity G' by taking all the granules of G whose labels are between m and n. For example, given granularity `year`, all the years in the 20th century can be generated by

$$\texttt{20CenturyYear} = Subset^{1999}_{1900}(\texttt{year}).$$

Note that G' is a label-aligned sub-granularity of G, and G' is not a full-integer labeled granularity even if G is. We also allow the extensions of setting $m = -\infty$ or $n = \infty$ with semantics properly extended.

Selecting operations The selecting operations are all binary operations. They generate new granularities by selecting granules from the first operand in terms of their relationship with the granules of the second operand. The result is always a label-aligned sub-granularity of the first operand granularity.

There are three selecting operations: *select-down, select-up* and *select-by-intersect.*

Select-down operation. For each granule $G_2(i)$, there exits a set of granules of G_1 that is contained in $G_2(i)$. The operation $Select\text{-}down^l_k(G_1, G_2)$, where $k \neq 0$ and $l > 0$ are integers, selects granules of G_1 by selecting l granules starting from the k^{th} one in each set of granules of G_1 that are contained in one granule of G_2. For example, Thanksgiving days are the 4^{th} Thursdays of all Novembers. If granularities `Thursday` and `November` are given, it can be generated by

$$\texttt{Thanksgiving} = Select\text{-}down^1_4(\texttt{Thursday}, \texttt{November}).$$

Note that G' is a label-aligned sub-granularity of G_1.

Select-up operation. The select-up operation *Select-up*(G_1, G_2) generates a new granularity G' by selecting the granules of G_1 that contain one or more granules of G_2. For example, given granularities `week` and `Thanksgiving`, the weeks that contain Thanksgiving days can be defined by

$$\texttt{ThanxWeek} = \textit{Select-up}(\texttt{week}, \texttt{Thanksgiving})$$

Note that G' is a label-aligned sub-granularity of G_1.

Select-by-intersect operation. For each granule $G_2(i)$, there may exist a set of granules of G_1 each intersecting $G_2(i)$. The operation *Select-by-intersect*$_k^l(G_1, G_2)$, where $k \neq 0$ and $l > 0$ are integers, selects granules of G_1 by selecting l granules starting from the k^{th} one in all such sets, generating a new granularity G'. For example, given granularities `week` and `month`, the granularity consisting of the first week of each month (among all weeks intersecting the month) can be generated by

$$\texttt{FirstWeekOfMonth} = \textit{Select-by-intersect}_1^1(\texttt{week}, \texttt{month}).$$

Again, G' is a label-aligned sub-granularity of G_1.

Set operations The set operations are based on the viewpoint that each granularity is a set of granules. In order to have the set operations as a part of the calendar algebra and to make certain computations easier, we restrict the operand granularities participating in the set operations so that the result of the operation is always a valid granularity: *The set operations can be defined on G_1 and G_2 only if there exists a granularity H such that G_1 and G_2 are both label-aligned sub-granularities of H.* In the following, we describe the union, intersection and difference operations of G_1 and G_2, assuming that they satisfy the requirement.

Union. The union operation $G_1 \cup G_2$ generates a new granularity G' by collecting all the granules from both G_1 and G_2. For example, given granularities `Sunday` and `Saturday`, the granularity of the weekend days can be generated by

$$\texttt{WeekendDay} = \texttt{Sunday} \cup \texttt{Saturday}.$$

Note that G_1 and G_2 are label-aligned sub-granularities of G'. In addition, if G_1 and G_2 are label-aligned sub-granularity of H, then G' is also a label-aligned sub-granularity of H. This can be seen from the transitivity of the label-aligned sub-granularity relationship (proof is left to the reader).

Intersection and difference operations can be similarly defined.

Syntactic restrictions on algebra operations

The granularities participating in a calendar operation usually have to satisfy certain conditions. For example, the set operations only apply to granularities that are label-aligned sub-granularities of a common one. Checking these preconditions can be difficult.

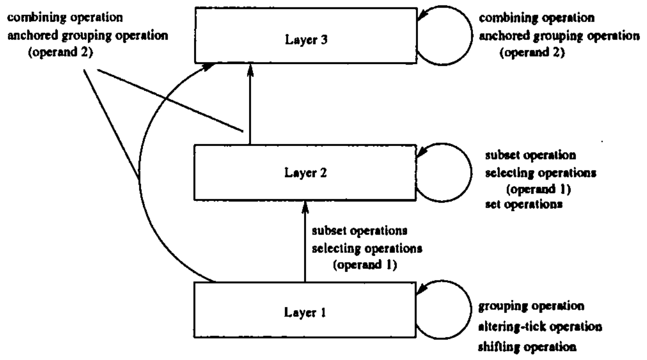

Figure 2: Transition between the three layers

Our solution is to use a syntactic restriction, namely to use the explicit relationship derived from the operations themselves. Note that the preconditions of the operations only use the following kinds of requirements: (1) a granularity must be a full-integer labeled one, (2) a granularity must partition another one, and (3) a granularity is a label-aligned sub-granularity of another.

The above syntactic restriction actually gives a classification of the granularities that can be generated from the calendar algebra. The granularities can be seen as organized into three layers. The membership of a granularity in a layer is determined by the operations and the operands used to define it. Fig. 2 shows the three-layered partition of the granularities defined by the calendar algebra and the transitions between the layers resulting from calendar algebraic operations. *Layer 1* consists of the bottom granularity and the granularities generated by only applying (may be repeatedly) the basic operations (grouping, altering-tick and shifting). *Layer 2* consists of the granularities that are the result of applying (may be repeatedly) the subset operation and the selecting operations on the full-integer labeled granularities in the first layer. Note that the second operand used in the selecting operations can be a granularity in any layer. *Layer 3* consists of granularities that are the result of the combining and anchored grouping operations. Note that operand 1 for the anchored grouping operation must be from layer 1 (a full-integer labeled granularity), while the combine operation may take granularities of any layers.

Granularities in the three layers have distinct properties. All the granularities in layer 1 are full-integer labeled granularities. All granularities in layer 2 may not be full-integer labeled ones, but there is no gap within each granule of every granularity, i.e., each granule is an interval of granules of the bottom granularity. The granularities in layer 3, however, may contain gaps within a granule.

Examples

In this subsection, we present some more example granularities represented by the calendar algebra. We assume second is the bottom granularity. Then we may have the following.

- minute = Group_{60}(second),
- hour = Group_{60}(minute),
- day = Group_{24}(hour),
- week = Group_{7}(day),
- pseudomonth = $\mathit{Alter}^{12}_{11,-1}$(day, $\mathit{Alter}^{12}_{9,-1}$(day, $\mathit{Alter}^{12}_{6,-1}$(day, $\mathit{Alter}^{12}_{4,-1}$(day, $\mathit{Alter}^{12}_{2,-3}$(day, Group_{31}(day)))))), where pseudomonth is generated by grouping 31 days, and then shrink April (4), June (6), September (9) and November (11) by one day, and shrink February (2) by 3 days,
- month = $\mathit{Alter}^{12*400}_{2+12*399,1}$(day, $\mathit{Alter}^{12*100}_{2+12*99,-1}$(day, $\mathit{Alter}^{12*4}_{2+12*3,1}$(day, pseudomonth))), where the February of each leap year is adjusted appropriately,
- Monday = $\mathit{Select\text{-}down}^{1}_{1}$(day, week),

  ...
- Sunday = $\mathit{Select\text{-}down}^{1}_{7}$(day, week).

In the above examples, we assumed that second(1) starts a minute, minute(1) starts an hour, etc. These are actually realistic for the Gregorian calendar.

To study the expressiveness of the calendar algebra, we define the concepts of periodical granularity and finite granularity. A G granularity is said to be *periodical* (wrt the bottom granularity B) if there exist positive integers R and P, where R is less than the number of granules of G, such that for each integer i, if $G(i) = \bigcup_{r=0}^{k} B(j_r)$ and $G(i+R) \neq \emptyset$, then $G(i+R) = \bigcup_{r=0}^{k} B(j_r + P)$. Assume B = day. It is easily seen that week is periodical with $R = 1$ and $P = 7$. Furthermore, year is also periodical with $R = 400$ and P being the number of days in a four-hundred year period. A granularity is said to be *finite* if the number of its granules is finite.

Theorem The calendar algebra can represent all the granularities which are either periodical or finite.

In addition to the periodical and finite granularities, the calendar algebra can also represent other granularities. In order to avoid certain granularities, we may disallow certain operations and restrict the use of operations in certain orders. We omit the details here.

Labeling the granules

The mapping viewpoint of granularity reveals the nature of the time granularities and time can be processed based on the indexes of the granularities, which are countable and easy to process with computers. However, human users are used to *relative and textual representation* of granules. Although label mappings for formally defined granularities were suggested(BDE+98) so that granules may have textual representations, no formal framework has been proposed before.

In this section, we introduce the notion of a *label* to address this issue, realizing and extending the idea of a label mapping. We assume that all the "words" used in the textual representation of granules are enumerable, so they can be encoded as integers. For example, the English words for months, i.e., January, February, March, ..., and December, can be encoded as integers 1 to 12. As a result, we need only deal with integers for the textual representation.

Definition (labels) Given a granularity G, let $G_k, G_{k-1}, \ldots, G_1$ be a sequence of granularities, where G is a label-aligned sub-granularity of G_1. A vector $(j_k, j_{k-1}, \ldots, j_1)$ in N^k is said to be a *label* of the granule $G(i)$ if $G(i) = G_1(j_1')$ is the j_1^{th} granule of all the granules of G_1 that overlap the granule $G_2(j_2')$, which is the j_2^{th} granule of all the granules of G_2 that .... If any of the above j_l^{th} granule does not exist, then the above vector is not a label for any granule.

As an example, consider the granularity Sunday which is label-aligned subgranularity of Sunday. Consider the granularity sequence year, month, Sunday. Then (1998, 7, 2) corresponds to *the second Sunday in July 1998*. Also, Sunday is label-aligned subgranularity of day, and consider year, month, day. The vector (1998, 5, 3) (i.e., May 3rd, 1998), which is the first Sunday in May 1998, is a label for granularity Sunday. However, the vector (1998, 5, 4) (i.e., May 4, 1998, which is a Monday) is not a label, since it cannot be mapped to any index of Sunday.

Granule Conversion

Granularity represents the unit of measurement for temporal data. In order to process data measured in different granularities, systems should have the ability to convert the granules in one granularity to those in another. For example, suppose a database stores daily sales information. To get the sales data per business month, the database application has to have the information that which day is in which business month. We refer to the process of finding some granules in one granularity in terms of the granules in another as *granule conversion.*

There can be many different semantics for granule conversions. In this section, we propose a generic conversion method on the basis of calendar regardless of the semantics of the conversion. Our method is based on three basic constructs, up conversion, down conversion and next conversion (which is a conversion within one granularity). We also identify an important class of semantics of granule conversion, and demonstrate an example for a general conversion with our method.

The generic method

In a given calendar, one granularity groups into another[2] if the latter is "defined only on" the former. (Granularity G is "defined only on" granularity H if

[2] H groups into G if each granule of the G is exactly the union of some granules of H (BDE+98).

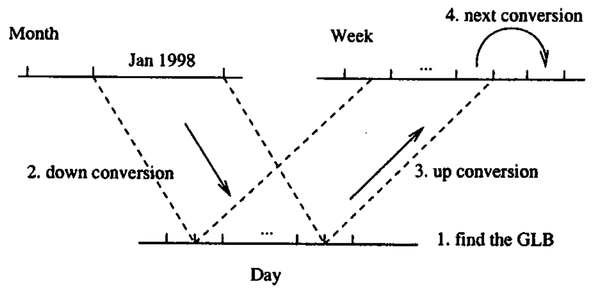

Figure 3: A conversion between *Month* and *Week*

in the calendar, G are obtained starting from H using algebra operations.) Hence, for each granule of the latter, there exist a set of granules of the former such that both sets of granules cover the same part of the time domain. Clearly, the greatest lower bound with respect to defined only on relation (*GLB*) always exists. Therefore, conversion between two granularities can be done with their *GLB* as an intermediary. (In the worst case, the bottom granularity will be this intermediary.)

In following, we present the three basic conversions.

Definition (Down conversion) Let G and H be granularities, where G is defined only on H. *Down conversion* from G to H, denoted $\lfloor\cdot\rfloor_H^G$, is a mapping from the index set of G to the subsets of the index set of H such that for each index i of G, the down conversion $\lfloor i\rfloor_H^G$ consists of all and only the indexes of the granules of H that group into $G(i)$, i.e., $G(i) = \cup_{j\in\lfloor i\rfloor_H^G} H(j)$.

Definition (Up conversion) Let G, H be granularities, where G is defined only on H. *Up conversion* from H to G, denoted $\lceil\cdot\rceil_H^G$, is a mapping from the index set of H to the index set of G such that for each index i of H, if there exists a granule $G(j)$ that contains $H(i)$, then $\lceil i\rceil_H^G = j$; otherwise $\lceil i\rceil_H^G$ is undefined.

When granualrity G is defined only on granularity H in a calendar, up and down conversion represent two directions of the conversions between G and H. Down conversion from G to H gets the indexes of the granules of H that group into a certain granule of G, representing the *down* direction, while up conversion from H to G gets the index to the granule that contains a certain granule of H, representing the *up* direction.

Using the granularity examples given earlier, we can see that the down conversion from `month` to `day` is a mapping such that $\lfloor 1\rfloor_{\mathtt{day}}^{\mathtt{month}} = \{1,2,...,31\}$, $\lfloor 2\rfloor_{\mathtt{day}}^{\mathtt{month}} = \{32,33,...,59\}$, etc. While the up conversion from `day` to `month` is a mapping such that $\lceil 1\rceil_{\mathtt{day}}^{\mathtt{month}} = \lceil 2\rceil_{\mathtt{day}}^{\mathtt{month}} = ... = \lceil 31\rceil_{\mathtt{day}}^{\mathtt{month}} = 1$, $\lceil 32\rceil_{\mathtt{day}}^{\mathtt{month}} = \lceil 33\rceil_{\mathtt{day}}^{\mathtt{month}} = ... = \lceil 59\rceil_{\mathtt{day}}^{\mathtt{month}} = 2$, etc.

Definition (Next conversion) Let G be a granularity. *Next conversion* within granularity G, denoted $Next_G(\cdot)$, is a mapping from $Z \times Z$ to the index set of G such that for each pair (i,n)

- if $n > 0$ and there exists a granule $G(j)$ which is the nth granule of G whose index is greater than i, let $Next_G(i,n) = j$;
- if $n < 0$ and there exists a granule $G(j)$ which is the $|n|$th granule of G whose index is less than i, let $Next_G(i,n) = j$;
- if $n = 0$ and $G(i)$ is a granule of G, let $Next_G(i,0)=i$;
- otherwise let $Next_G(i)$ be undefined.

General conversions

With down, up and next conversion, a general purpose conversion can be performed with further consideration of the conversion semantics. Let G_1 and G_2 be granularities involved in a granularity conversion problem. The first step of the conversion would be to find the *GLB* of G_1 and G_2 in the calendar. Let granularity H be the *GLB* of them. With H as the intermediary, an appropriate set of granules of G_1 will be converted to H by down conversion. Then a corresponding set of granules of G_2 can be found by up conversion. Finally, the conversion problem is solved by the combination of up, down and next conversion under the conversion semantics.

For example, suppose we want to know the second week after `January 1998` (in `month`). As the first step, we find that their *GLB* in the calendar is granularity `day`. So we use `day` as the intermediary for this conversion. As the second step, the `day` that group into `January 1998` are collected by down conversion from `month` to `day`. As the third step, the `week` containing the last `day` of `January 1998` is found by an up conversion from `day` to `week`. Finally, the `second week` after `January 1998` is computed with a next conversion.

Three conversion semantics

Granule conversion is essential in applications related to time, such as automatic evaluation of user queries involving multiple granularities, rolling up or drilling down along time hierarchy in OLAP applications and time series analysis. Some conversion semantics are very frequently used in these applications. For example, when rolling up along a time hierarchy in an OLAP or time series analysis application, the application must know the relationship about how granules of a finer granularity are contained in granules of a coarser granularity in order to fulfil the analysis. When drilling down a time hierarchy to estimate how data of a coarser granularity is distributed in the granules of a finer granularity, say estimate the daily sales according to the stored monthly sales, the application must use a similar relationship again in addition to the assumptions about the distribution.

We abstract these conversion semantics into the following three categories:

- *Covering.* The granularity conversion should return all the granules of the destination granularity such that the time represented by the source granules contains the time represented by each destination granule;

- *Covered-by.* The granularity conversion should return the smallest set of granules of the destination granularity such that the time represented by the source granules are convered by the time represented by the destination granules.
- *Overlap.* The granularity conversion should return all and only the granules of the destination granularity such that the time represented by the source granules overlaps the time of each destination granule.

Computation of down and up conversions

As discussed earlier, the computation of the up and down conversions is very critical. Because of the index manipulation nature of the calendar algebraic operations, the up and down conversion can be recursively computed.

It's been discussed that the granularities in a calendar can be divided into 3 layers. In layer 1, all the granularities are full-integer labelled granularities. There exist simple formulas for the up conversion and the down conversion for the shifting operation and the grouping operation. Though the altering granule operation is a bit more complex, there also exists simple formula for the down conversion, and the up conversion can be done on the basis of the down conversion. Furthermore, the up conversion for the altering tick operation can be estimated, and the difference between the estimated value and the real up conversion is usually bounded by a small number. Suppose the number of basic operations that are involved in the conversion is n. The complexity of the up and the down conversion in layer 1 is linear to n if there is no up conversion for the altering tick operation. If there exists up conversion for the altering tick operation, the complexity is $O(n \cdot log_2 P)$ in the worst case, where P is the number of granules in one period. However, we usually have near linear algorithm if the number of finer granularity granules in the granules of the coaser granularity don't vary very much. Therefore, there exist efficient algorithms for the up and down conversions in the first layer.

Consider layers 2 and 3 of the calendar. In general, the algorithm for the up and the down conversion are not only affected by the number of operations involved, but also by the correspondence of the granules of both operand granularities, e.g., how many granules of the first operand are contained in the granules of the second operand in select-down operation. Because the indexes of a second-layer or third-layer granularity are not assumed to be contiguous, the conversion has to individually manipulate the indexes. However, with further knowledge, e.g., one or both operands are in layer 1 or layer 2, the conversion algorithm can be more efficient. If both operands of the selecting operations or the combining operaton are not in the third layer, each operand granularity doesn't have inside gaps, so the processing of the coarser granularity is simplified. If it is further known that the finer operand is in layer 1, the fact that the indexes of the finer operand are contiguous can be utilized, and the complexity of the conversion is only related to the number of operations involved.

Computation of next conversion

Next conversion is trival for layer 1 granularities because of the contiguity of their indexes ($Next_G(i, n) = i + n$). However, it can be a difficult problem for layer 2 and 3 granularities, where indexes may not be contiguous any more.

A desirable solution would be getting the result with the information of the operations. For example, suppose there is a granularity which stands for the first day of every month. To get the nth granule after a base granule, say i, we only need to get the nth month after the month containing granule i, and finding the result would be easy. In this case, the next conversion for a granularity is translated into a trival one for a granularity. However, we couldn't find a general algorithm to get the result in this way. In this abstract, we will outline alternative ways. There are two straightforward ways to solve this problem in addition to making use of the information of the operations.

1. Search for the nth granule by testing. The basic construct is to determine whether an integer is a valid index or not. To get the result, the algorithm tests the integers one by one until the nth valid granule is found.
2. Enumerate the valid indexes. This involves precomputation of valid indexes and storing them. Since all the granularities are periodical (see earlier definition or (BDE$^+$98)), the valid indexes must be periodic. So enumeration of one period is enough.

Obviously, the first method is only suitable for granularity with dense indexes and small n. In other cases, it will result in unacceptable performance. Although the second method has good computation performance, it doesn't scale well when the period gets big. In the following, we propose several enhancements that can improve the scalability, and sometimes make a trade off between the computational efficiency and the storage requirement to get overall performance.

Our first enhancement is to use a hash table to maintain the valid index information for a granularity. We distinguish two kinds of hash tables, the first one is a ***positive*** hash table, in which valid indexes within a period are stored, and the second one is a ***negative*** hash table, in which missing indexes (i.e. the integers that are not valid indexes) are stored. It is easy to see that the positive hash table and the negative hash table are complementary. The positive hash table is used when the indexes are sparse, while the negative hash table is used when the indexes are dense. The hash based method is suitable for granularities with both dense and sparse index. The distinction of positive and negative hash tables reduces the size of the enumeration by at least a half. However, this may not solve the scalability problem. Nevertheless, we can improve the scalablity by sacraficing some computational efficiency. If the hash table gets too big, we can reduce the size by only storing a part of valid indexes. It would be desirable that the densities of the valid indexes are almost the same between each consecutive hash entries.

An alternative enhancement is to use bitmap for the valid indexes of a granularity. Since the indexes are periodic, bitmap is only necessary for a period. Each bit in the bitmap corresponds to an integer. A bit is 1 if the corresponding integer is a valid index, 0 if not. Finding the nth granule after a base granule is just counting n 1 in the bitmap and finding the corresponding integer. The advange of such representation is: only generalzied granularities that are defined by select operations need precomputed bitmap. If a granularity is defined by a set operation, then the corresponding bitmap can be easily composed with the bitmaps of the operands. Suppose A and B are granularities with bitmaps a and b respectively. Then the bitmaps for $A \cup B$, $A \cap B$ and $A - B$ are a OR b, a AND b and a AND (NOT b) respectively. To save the space, compression method, e.g. run length encoding, can be utilized, which can make the counting of 1s even faster. Although bitmap method can save space for granularities defined by set operations, it may not scale well for those with long periods.

Related Work

Much work has been done on the problem of granularity representation in temporal database area as well as other areas like artificial intelligence and real time systems. Some of them address the formalization of time granularity systems (CR87; Dea89; MMCR92; BWJ98). Our work is an instantiation of the general framework proposed in (BWJ98).

MultiCal project (SS95) and TSQL2 (Sno95) are both language extensions to Structured Query Language. Irregular mappings (granules can not be converted by a simple multiply or divide), e.g. between *month* and *day*, have to be specified by a piece of program, e.g. a C function (Sno95; Lin97). Our representation improves this method by providing a set of calendar operations to define the granularities in a declarative way.

A representation of granularities that allows natural language expression was proposed in (LMF86) on the basis of structured collections of intervals. This representation was later implemented in *POSTGRES* (CSS94). As the foundation of the system, the primitive collections, e.g. *day*, *month*, *year*, have to be enumerated, though there exist some patterns in them. Our work does not require explicit enumeration.

There are also other proposals for granularity representation. In (LRW96), a granularity (called calendar in (LRW96)) is modeled as a totally ordered set of intervals with additional semantics, and several calendar operations are introduced to generate user defined time granularities. A system defined granularity is formed by the relative pattern of its granules with respect to the granules of another granularity. Similar to the primitive collections in (LMF86), this is basically enumeration. Our approach can achieve the same results with a subset of operations without enumeration.

Conclusion

We proposed an algebraic representation of calendars that favors the inter-granularity relationship. The formalization of the calendar turns out be useful in both granularity conversion and the formal construction of the labeling schemes. In addition, labels present a way of constructing naming conventions for the granularities in the calendar. As continuance of (BWJ98), this work provides a more specific multiple granularity support for time related applications.

References

C. Bettini, C.E. Dyreson, W.S. Evans, R.R. Snodgrass, and X. S. Wang. *Temporal Databases: Research and Practice*, volume 1399 of *Lecture Notes in Computer Science*, chapter A Glossary of Time Granularity Concepts. Springer, 1998.

C. Bettini, X.S. Wang, and S. Jajodia. A general framework for time granularity and its application to temporal reasoning. *Annals ofMathematics and Artificial Intelligence*, 22(1-2):29–58, 1998.

J. Clifford and A. Rao. A simple, general structure for temporal domains. In *proc. of the Conference on Temporal Aspects in Information Systems*, pages 23–30, France, 1987.

R. Chandra, A. Segev, and M. Stonebraker. Implementing calendars and temporal rules in next generation databases. In *Proceedings of ICDE*, pages 264–273, 1994.

T. Dean. Using temporal hierarchies to efficiently maintain large temporal databases. *JACM*, 36:687–718, 1989.

H. Lin. Efficient conversion between temporal granularities. Technical Report 19, Time Center, July 1997.

B. Leban, D. McDonald, and D. Foster. A representation for collections of temporal intervals. In *Proceedings of AAAI*, pages 367–371, 1986.

J.Y. Lee, E. Ramez, and J. Won. Specification of calendars and time series for temporal databases. In *International Conference on the Entity Relationship Approach (ER)*, pages 341–356, 1996.

A. Montanari, E. Maim, E. Ciapessoni, and E. Ratto. Dealing with time granularity in the event caleulus. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, pages 702–712, Tokyo, Japan, 1992.

R.T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer Academic Pub., 1995.

M. Soo and R. Snodgrass. Mixed calendar query language susport for temporal constants (release 1.1). The MultiCal Project. Department of Computer Science, university of Arizona, September 1995.