RJ 10282 (A0302-007) February 13, 2003
Computer Science

IBM Research Report

A Framework for User Defined Periodic Calendars

Yael Shaham-Gafni, Daniel A. Ford
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

A Framework for User Defined Periodic Calendars

Yael Shaham-Gafni **Daniel Ford**

Abstract

Most contemporary calendaring and scheduling systems represent time using the Gregorian calendar. Systems that are more conscious of internationalization provide a choice of several predefined calendaring systems. We have observed that humans are constantly contriving new systems of time description that reflect specific domains of their activities, such as academic, agriculture, finance, sports calendars, and more. In this paper we investigate a model that offers the flexibility of adding an arbitrary time description system to an existing calendaring and scheduling application. Such a model allows organizations to define their own calendar and the periods of time that are significant to their activities, so that users within the organization can view, plan and reason about time in terms relevant to their domain.

Most of the formal symbolic definitions of calendars are logic oriented, and use constraints to represent valid dates, and to represent transformation rules between different calendars. This type of representation is not intuitive, and requires many hours of learning for a casual user to master. In this work we took an object oriented approach to representing calendars. The temporal concepts are represented by classes that are organized in hierarchies, and are related to each other via aggregation. A calendar definition is created by declaring instances of these classes, and the actual links between them.

1 Introduction

The problem of measuring and representing time is as old as human civilization. Virtually all successful (?) human civilizations created a calendaring system that allowed them to track the changes in their environment and to establish the times for various agricultural, religious or civil events that bonded their communities together. While each such calendar system is as unique as the culture that created it, they all have a single common feature, they are all periodic. This trait is no accident, the only accurate temporal reference available to ancient cultures were the orbits of celestial bodies such as the Earth, the Moon and Venus. These orbits are periodic.

In this paper we describe a model and representation for declaratively defining arbitrary periodic calendar systems. This model was specifically created to allow the definition of new specialized (periodic) calendar systems that have domain specific uses. While it has modest goals, it turns out that the model is powerful enough to describe most historical human calendaring systems (e.g., Gregorian, Islamic, Mayan). A limitation of the model is that it cannot completely describe calendar systems that have arbitrary rules that require logical interpretation; the Hebrew calendar is an example of this type of system. But, given that the main purpose of the model is to define new calendar systems, not old ones, that limitation is of little consequence.

The development of this model is a direct result of the observation that human activities have become quite complex and specialized, and that humans are constantly contriving new systems of time description that reflect specific domains of their activities. The model is needed because the technology for supporting these new temporal inventions has not yet evolved. For instance, academic organizations known as "Universities" have a specific temporal measurement that they call the "Academic Year" and they even publish an "Academic Calendar" that specifies the various important subdivisions of that "year" (e.g., "Fall Semester", "Spring Break", "Winter Exam Period", "Summer Vacation", etc.). Various other examples exist in other domains (e.g., agriculture, finance, sports). The problem is that all of these domains are currently forced to map their temporal organizations (i.e., calendar systems) on to another calendar system (usually the Gregorian calendar system) that has measurement periods that are completely unrelated to the domain. Why should an academic calendar be mapped to arbitrary units of time dedicated to the memories of dead Roman emperors (e.g., Augusts) A more appropriate solution would be to allow new units of time to be defined that match the domain and allow them to be organized into a full system that more directly supports specific human activities.

The model and framework we have developed enable a user (an organization) to declaratively describe and define a periodic calendar. The framework supports integrating a multiple calendars, and provides means for graphically displaying a periodic calendar and customizing the display. Using a straightforward language, the user can define the basic time units, the relations between them, and what constitutes a date in the calendar. The framework knows how to interpret these definitions and can produce a graphic depiction of the desired calendar, or of multiple calendars simultaneously.

Our model is structured according to the object oriented paradigm and consists of a framework of classes. The temporal concepts are represented by classes that are organized in a hierarchy. Other classes are related to each other via aggregation. The user proceeds by declaring instances of these classes, and the actual links between them. Section 2, "A Model for Defining Periodic Calendar Systems, describes the temporal class hierarchy in detail, section 3An Example of a User Defined Calendar", shows how these definitions are used to declare a calendar, section 4 describes how the calendar system definitions are used to count time and create dates. In section 5, "Related Work", we survey other work in this area, and in section 6 we conclude.

2 A Model for Defining Periodic Calendar Systems

A calendar system typically defines a set of *time units*, each time unit is intended to measure time in a different level of granularity. The units are usually related, such that one unit consists of *n* repetitions of another unit. We capture the notion of a time unit in our *TemporalMeasure* class. Examples of TemporalMeasures are second, day, month, year, and century. Another purpose of the time units is to provide a labeling scheme for intervals of time, that covers the timeline, and allows to easily identify a time

interval. For example "January 1, 2000" is a label that identifies a specific day. We denote a labeled time interval as a *date*. The labeling scheme defines both form and content. The form of a label describes which time units participate in a date and the content of a label uniquely identifies a date. In the following subsections we describe the classes that capture these notions in detail.

2.1 Temporal Measure

A *TemporalMeasure* is an abstraction for measuring the length of time, i.e. a definition of a unit of time, e.g. second, day, decade. A TemporalMeasure therefore defines a time duration.

Formally a TemporalMeasure *T* consists of a *name* and a *time span*. In the most general case, the time span of a TemporalMeasure is *T* itself.

2.2 SubdividedTemporalMeasure

A TemporalMeasure can be divided into smaller units of time. Such a temporal measure is called a *SubdividedTemporalMeasure*. For example a minute is divided into 60 seconds. In this case we say that a second is a *subMeasure* of minute, and minute has one subMeasure which is repeated 60 times. A measure need not be divided into equal sized parts. For example a year is divided into 12 months but their lengths differ. In this case the measure year has a sequence of 12 subMeasures each having it's own length.

Formally, a SubdividedTemporalMeasure *T* is a TemporalMeasure that has a sequence of *subMeasures* (T_1, ... T_n) which are themselves TemporalMeasures, and a *repetition* which is a positive (>0) integer number. The time span of a SubdividedTemporalMeasure is the sum of the time spans of its subMeasures times the repetition. For example let *T, R, S,* be temporal measures, such that *(R, S, S)* are the subMeasures of *T*, and the repetition of *T* is *3*. Then the time span of T is *3*R + 6*S*.

2.3 PeriodicSubdividedTemporalMeasure

A SubdividedTemporalMeasure can define an infinite duration of time, by infinitely repeating its sequence of subMeasures. Such a temporal measure is called a *PeriodicSubdividedTemporalMeasure*. For example AD (anno Domini, the Christian era) is an infinite repetition of years.

Formally, a PeriodicSubdividedTemporalMeasure *T* is a SubdividedTemporalMeasure whose repetition is infinity (∞)

2.4 EquivalenceTemporalMeasure

Some TemporalMeasures are inexact, for example a year may be 365 days, or 366 days long, depending whether it is a leap year or not. In this situation the measure year represents a set of *equivalentMeasures*. Such a temporal measure is called an *EquivalenceTemporalMeasure*.

Formally, an EquivalenceTemporalMeasure *T* is a TemporalMeasure that has a set of *equivalentMeasures* (T_1, ... T_n) which are themselves TemporalMeasures. The time span of an EquivalenceTemporalMeasure remains as for a TemporalMeasure, *T* itself.

2.5 LabelingTemporalMeasure

Up to this point we have shown how TemporalMeasures define time durations and subdivide time into various durations. Our goal is to generate labels for these durations such that the timeline can be covered. Intuitively, every time unit has a set of values that is associated with it. For example in the Gregorian calendar a year is associated with positive integer values: {1, 2, ..., 1999, 2000, ...} and a month is associated with a name out of the set { January, ..., December }.

An important question that comes up is: who has the 'knowledge' to provide a label for the next time interval of a time unit. This question is best illustrated through an example. Lets consider two schemes for labeling days: The day of the week scheme, and the day of the month scheme. In the day of the week scheme, there are seven names that are repeated cyclically: {Monday, ... , Sunday}. In this case we say that the 'day' time unit has enough knowledge to generate names for itself, given the set of weekday names. In the day of the month scheme, there are 31 names taken from the integer set { 1, ... , 31 }, but not all names are used in every month, thus the choice of day names depends on which month it is. In this case the 'day' time unit does not have enough knowledge to generate names for itself, only the containing time unit, month, has that knowledge. In terms of our ontology, a TemporalMeasure may generate labels for itself, and a SubdividedTemporalMeasure may in addition generate labels for its subMeasures.

A *LabelingTemporalMeasure* is a TemporalMeasure that has a *time-value set* associated with it. A time-value set is a linearly ordered set of labels, which can be finite or infinite. Every time the LabelingTemporalMeasure is asked for a label, it returns the next label in the time-value set. If the set is finite, the labels are repeated cyclically.

Formally, a LabelingTemporalMeasure *T* is a TemporalMeasure that has *time-value set V* which is a linearly ordered set of labels $(l_1, ..., l_n)$ (or $(l_1, ...)$ in case it is infinite), a *current value* which is one of the labels in the time-value set, and an operation *next(label),* that returns the next label according to the linear order, or the first label when the argument is the last label.

2.6 LabelingSubdividedTemporalMeasure

A *LabelingSubdividedTemporalMeasure* inherits from both LabelingTemporalMeasure and SubdividedTemporalMeasure. In addition, a LabelingSubdividedTemporalMeasure may generate labels for its subMeasures, by defining a *subMeasures-time-value set*. A subMeasures-time-value set, is a linearly ordered set of labels, which can be finite or infinite. Every time the LabelingSubdividedTemporalMeasure is asked for a label for a subMeasure, it returns the next label in the time-value set. If the set is finite, the labels are repeated cyclically.

Formally, a LabelingSubdividedTemporalMeasure *T* is a SubdividedTemporalMeasure and a LabelingTemporalMeasure that has *subMeasures-time-value set V* which is a linearly ordered set of labels $(l_1, ... l_n)$ (or $(l_1, ...)$ in case it is infinite), a *subMeasures current value* which is one of the labels in the time-value set, and an operation *subMeasures-next(label),* that returns the next label according to the linear order, or the first label when the argument is the last label.

2.7 LabeledTemporalMeasure

A *LabeledTemporalMeasure* is a TemporalMeasure who's labels are provided by a LabelingSubdividedTemporalMeasure.

Formally, a LabeledTemporalMeasure *T* is a TemporalMeasure that has a set of *labelingMeasures* $(T_1, ... T_n)$ which are themselves LabelingSubdividedTemporalMeasures.

2.8 Creating Additional Subclasses With Multiple Inheritance

Up until now we have described the basic subclasses of TemporalMeasure:

- SubdividedTemporalMeasure
- EquivalenceTemporalMeasure
- LabelingTemporalMeasure

We have also defined some specific subclasses: PeriodicSubdividedTemporalMeasure and LabelingSubdividedTemporalMeasure. Using multiple inheritance we can create additional subclasses. The characteristic of being subdividing or equivalence is mutually exclusive, a TemporalMeasure can be

only one of them. The characteristic of being labeling or labeled can be mixed and matched. Thus we can create additional subclasses from the basic subclasses:

- LabelingLabeledSubdividedTemporalMeasure
- LabelingEquivalenceTemporalMeasure

2.9 DesignationFormat

Most calendars identify a time interval by labeling it, and relating it to a higher level time interval that contains it. For example, in the Gregorian calendar, a day is labeled by the sequence: year, month, day. A *DesignationFormat* captures this notion.

Formally, a DesignationFormat *F* consists of a sequence of *designatingMeasures* $(T_1, ..., T_n)$, such that T_i is a LabelingTemporalMeasure and the duration of T_{i+1} is contained in T_i.

2.10 AbstractDate

An *AbstractDate* uniquely labels an interval of time according to a calendar.

Formally, an AbstractDate consists of a sequence of pairs (T_i, v_i) $i=1..n$, such that the following holds:

- The T_i are LabelingTemporalMeasures, and v_i is an element of the time-value set of T_i.
- $(T_1, ..., T_n)$ are the designatingMeasures, or a prefix of the designatingMeasures of a DesignationFormat.

2.11 CalendarSystem

The framework class that represents a calendar is *CalendarSystem*. A calendar system consists of:

- A set of TemporalMeasures, that are usually related to each other by subdivision or equivalence relations.
- A *top level measure*, which is the measure corresponding with the time interval the calendar represents. This can be a finite time interval, or an infinite time interval.
- A DesignationFormat, which describes how to form dates according to this calendar system.
- An epoch, which is a number relating the first day of this calendar to the Julian Day calendar. We chose the Julian Day calendar as our calendar of reference for reasons of convenience.

3 AAn Example of a User Defined Calendar

Our example is a simple academic calendar. Imagine a hypothetical university academic year that has three trimesters, each 14 weeks long. Between the trimesters there is a two week break period, and after the third trimester there is a vacation period of 6 weeks. All together the academic year is 3*14+2*2+6 = 52 weeks long which is 364 days. This academic year does not coincide with the civil year since it is one or two days shorter. What are the time units in this definition? The first obvious unit is the academic year. The year is subdivided into periods of various lengths: trimesters which are 14 weeks long, breaks which are 2 weeks long, and a vacation which is 6 weeks long. Thus the second unit will be called an academic period, and it has 3 different values, or equivalent time units. The third time unit is the week, and the last is a day. The definition of the time units for such a calendar follows in figure 1. Now that we have defined the time units of the system, let's concentrate on defining a date according to the simple academic year calendar. In order to reference a specific day we would need to know the year, period (trimester, break, vacation), week, and day of the week. The definition of a DesignationFormat for the simple academic calendar is shown in figure 2. The graphic presentation of this defined calendar is shown in Figures 3 - 5.

```
<PeriodicSubdividingTemporalMeasure>
    <name>Academic Era</name>
    <subMeasures>
        <li>Academic Year</li>
    </subMeasures>
</PeriodicSubdividingTemporalMeasure>

<LabelingSubdividingTemporalMeasure>
    <name>Academic Year</name>
    <timeValueSet>I, II, III, ...
    </timeValueSet>
    <subMeasures>
        <li>Trimester</li>
        <li>Break</li>
        <li>Trimester</li>
        <li>Break</li>
        <li>Trimester</li>
        <li>Vacation</li>
    </subMeasures>
    <repetition> 1 </repetition>
</LabelingSubdividingTemporalMeasure>

<LabelingEquivalenceTemporalMeasure>
    <name>AcademicPeriod</name>
    <timeValueSet>
        <li>Fall trimester</li>
        <li>Winter break</li>
        <li>Winter trimester</li>
        <li>Spring break</li>
        <li>Spring trimester</li>
        <li>Summer vacation</li>
    </timeValueSet>
    <equivalentMeasures>
        <li>Trimester</li>
        <li>Break</li>
        <li>Vacation</li>
    </equivalentMeasures>
</LabelingEquivalenceTemporalMeasure>

<LabelingSubdividingTemporalMeasure>
    <name>Trimester</name>
    <subMeasuresTimeValueSet>
        <li>Week 1</li>
        ...
        <li>Week 14</li>
    </subMeasuresTimeValueSet>
    <subMeasures>
        <li>Week</li>
    </subMeasures>
    <repetition> 14 </repetition>
</LabelingSubdividingTemporalMeasure>

<LabelingSubdividingTemporalMeasure>
    <name>Break</name>
    <subMeasuresTimeValueSet>
        <li>Week 1</li>
        <li>Week 2</li>
    </subMeasuresTimeValueSet>
    <subMeasures>
        <li>Week</li>
    </subMeasures>
    <repetition> 2 </repetition>
</LabelingSubdividingTemporalMeasure>

<LabelingSubdividingTemporalMeasure>
    <name>Vacation</name>
    <subMeasuresTimeValueSet>
        <li>Week 1</li>
        ...
        <li>Week 6</li>
    </subMeasuresTimeValueSet>
    <subMeasures>
        <li>Week</li>
    </subMeasures>
    <repetition> 6 </repetition>
</LabelingSubdividingTemporalMeasure>

<LabeledSubdividingTemporalMeasure>
    <name>Week</name>
    <labelingMeasures>
        <li>Trimester</li>
        <li>Break</li>
        <li>Vacation</li>
    </labelingMeasures>
    <subMeasures>
        <li>Day</li>
    </subMeasures>
    <repetition> 7 </repetition>
</LabeledSubdividingTemporalMeasure>

<LabelingTemporalMeasure>
    <name>Day</name>
    <timeValueSet>
        <li>Monday</li>
        <li>Tuesday</li>
        <li>Wednesday</li>
        <li>Thursday</li>
        <li>Friday</li>
        <li>Saturday</li>
        <li>Sunday</li>
    </timeValueSet>
</LabelingTemporalMeasure>
```

Figure 2: Simple Academic Year Time Units

```
<DesignationFormat>
    <name>Break</name>
    <designatingMeasures>
        <li>Day</li>
        <li>Week</li>
        <li>AcademicPeriod</li>
        <li>Academic Year</li>
    </designatingMeasures>
</DesignationFormat>
```

Figure 1: Simple Academic Year DesignationFormat

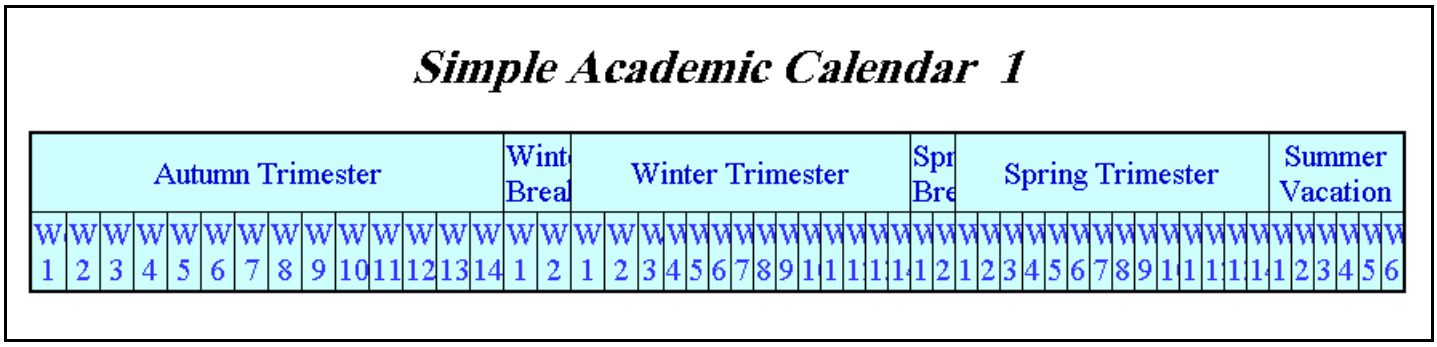

Figure 3: Simple Academic Calendar Graphic Display for 'Year'

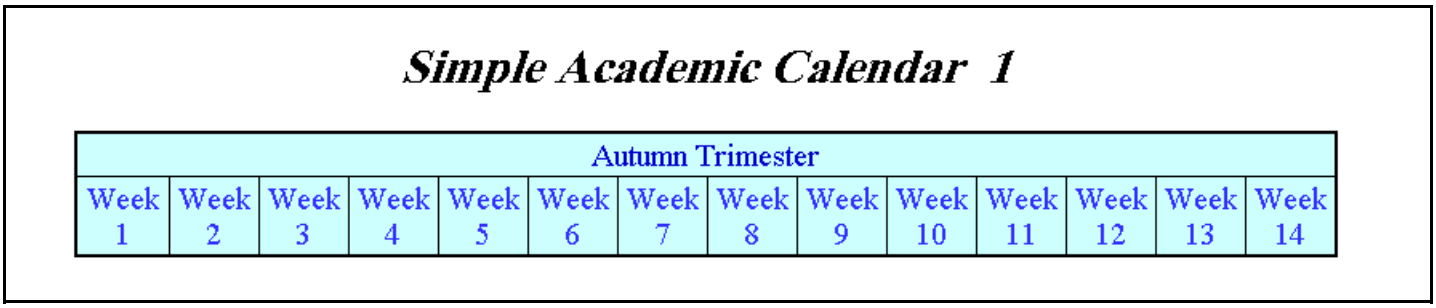

Figure 4: Simple Academic Calendar Graphic Display for 'Period'

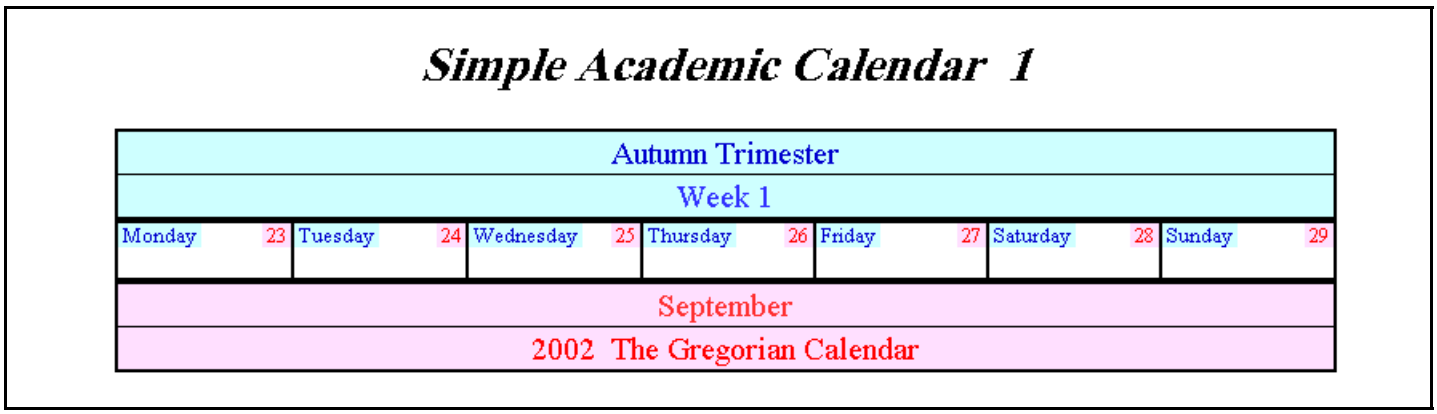

Figure 5: Simple Academic Calendar Graphic Display for 'Week'

4 Counting Time and Labeling Time Intervals

In section 2 we defined the different types of TemporalMeasures and the relations between them. In this section we will see how TemporalMeasures are used to count time, and how AbstractDates can be generated by a network of TemporalMeasures, to create a coverage of the timeline.

4.1 Counting Time According to a CalendarSystem

We count time by *ticking*. A TemporalMeasure ticks when the time unit it measures has elapsed. A SubdividedTemporalMeasure *T* that has subMeasures $(T_1, \ldots T_n)$ and *k* repetitions ticks when it completes *k* repetitions of receiving a tick from each of it's sub measures, *n*k* ticks. An EquivalenceTemporalMeasure

T that has equivalentMeasures $(T_1, \ldots T_n)$ ticks when it receives a tick from one of it's equivalent measures. A LabelingTemporalMeasure sets it's current value to the next value, every time it receives a tick.

Imagine an odometer that displays the distance your car has driven. Every mile the rightmost digit advances to the next value, and when the rightmost position has completed a full cycle, the digit to the left advances to its next value. The ticks propagate through the network of related temporal measures in a similar way. Keeping track of time according to a calendar system, when the time units are periodic and hierarchical, amounts to keeping track of the propagation of ticks. For this purpose we need to keep state information in the TemporalMeasures, and a stack. State information is kept for Subdividing TemporalMeasures, and consists of the current sub measure, and the current repetition. Before we begin counting time the stack needs to be initialized, and then counting time consists of updating the current state, and popping and pushing TemporalMeasures onto the stack according to the propagation of ticks. This is best illustrated through and example. Let's use the Simple Academic Calendar from section 3 to visualize how the ticking works. Figure 6(a) shows the stack and state for the last day of the Fall Trimester on Year I:
{ I, Fall Trimester, Week 14, Sunday}.

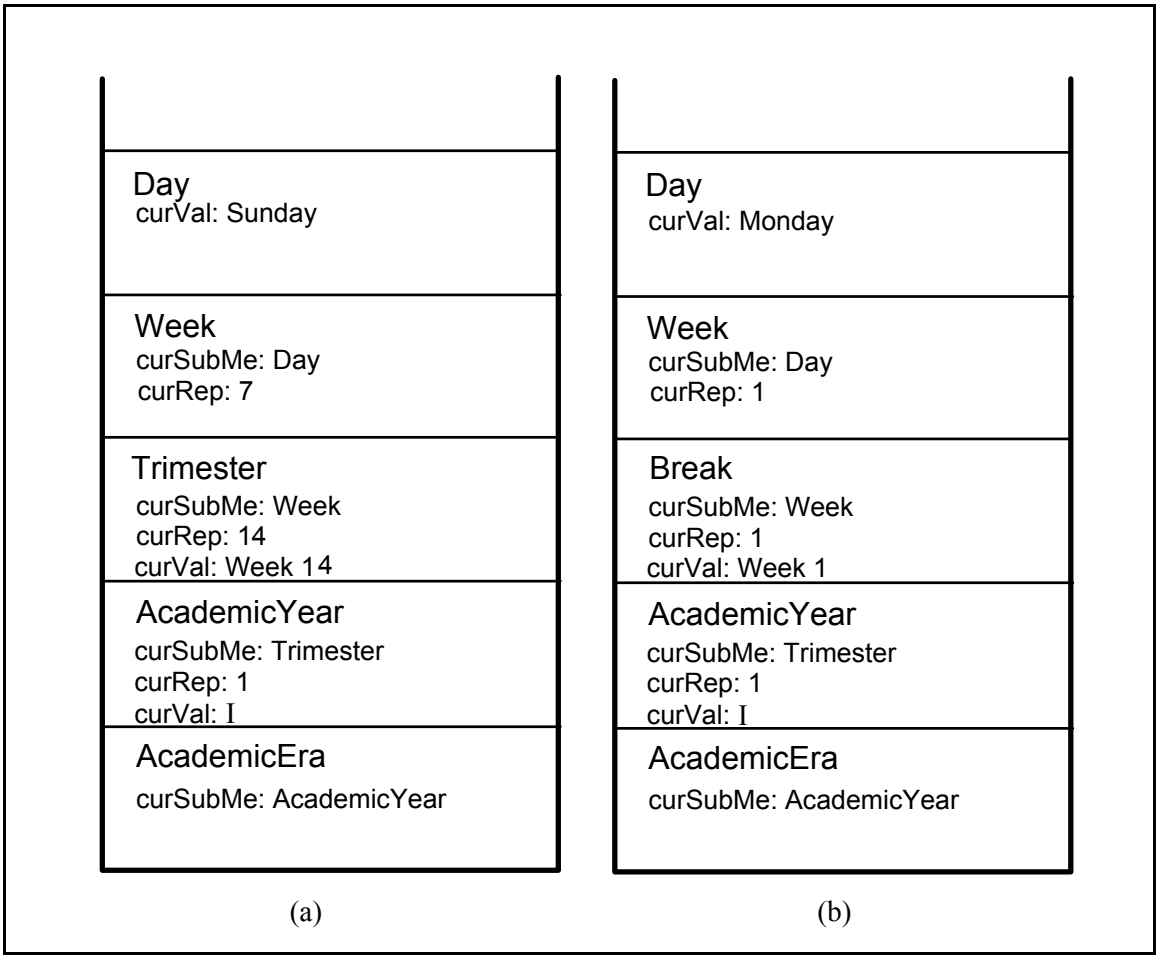

Figure 6: Example Stack and State Before and After Tick

Advancing the calendar to the next day results in the following actions:

- Day ticks, its current value is advanced to Monday and it is popped from the stack.
- Week receives a tick, since it has only one subMeasure it advances it's current repetition, and since it has completed a cycle, it ticks and is popped from the stack.

- Trimester receives a tick, since it has only one subMeasure it advances it's current repetition, and since it has completed a cycle, it ticks and is popped from the stack.
- Academic Year receives a tick, it advances it's current subMeasure to the next which is Break.
- A sequence of push operations on the stack follows where each TemporalMeasure that is on top of the stack pushes its current subMeasure. This results in the following pushes: push Break, push Week, push Day.

The resulting stack and state are shown in figure 6(b).

4.2 Generating Dates According to a CalendarSystem

In addition to providing a network of linked TemporalMeasures, a calendar system should specify how dates are to be formed. In section 2.9 "DesignationFormat" we described the framework class that is responsible for this. After every ticking step, the calendar's DesignationFormat is consulted to generate an AbstractDate that represents the date for that day according to the calendar system. The DesignationFormat for the Simple Academic Calendar example is shown in figure 2. In essence a date in the Simple Academic Calendar is formed by a list of values each belonging to a TemporalMeasure with a finer granularity, that is contained in the preceding TempoalMeasure. Figure 7 shows the values of the TemporalMeasures comprising the DesignationFormat for the date corresponding with the stack and state in figures 6(a).

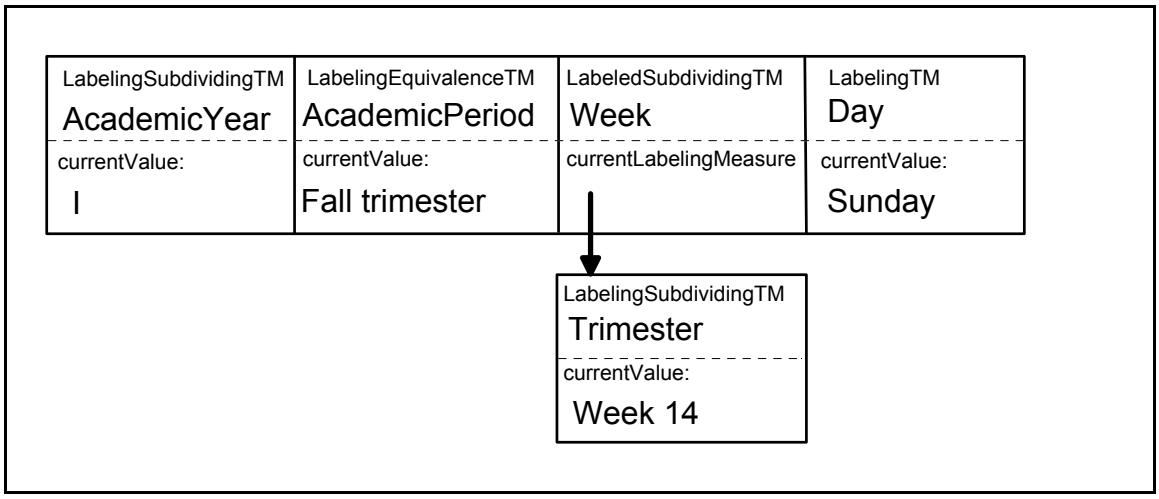

Figure 7: Example DesignationFormat with current values shown

5 Related Work

The calendaring and scheduling community is constantly working on improving and standardizing representations of time. The current standard, RFC2445 known as iCalendar [2] was developed by the IETF, and it's predecessor vCal [8] was developed by the Versit Consortium. FitzPatrick [5] presents a logical model, Orlando, for the representation of dates and times in a mnemonic yet machine readable format. The representation is in RDF schema based on iCalendar RFC2445. Once information is converted to this format, RDF Query languages can be used to ask intuitive questions about events data by taking slices through events along the axes of WHEN, WHERE and WHO. These representations are based on the Gregorian calendar, and do not support other calendaring systems.

In [10] Kraus, Sagiv and Subrahmanian provide a formal, theoretical definition of a calendar. The basic building block is a time unit, which is associated with a set of time values. The time units are arranged in a hierarchy, and the format of a date is determined by taking a path from the hierarchy root to a leaf. The

drawback of this definition is that it allows invalid dates, such as April 31, 2000. To overcome this, the authors introduce logic constraints to specify the valid dates of a calendar. In the same way, the merging of two calendars requires one to write the constraints that provide the mapping between the two calendars. Given these constrains the authors contend that the problem of the validity of a date is solvable using standard constraint solving techniques over finite domains and over the integers. Using constrains provide a powerful mechanism to describe any peculiarities of a calendar. It allows the description of calendars that are not necessarily periodic. On the other hand using standard constraint solving techniques seems impractical when one needs to quickly produce the consecutive dates of a month.

Ohlbach [14] presents a formal representation of time that is functional based. He defines the notion of a *Reference Time Line*, onto which all time representations are mapped. The building block of the time representation is called a *Time Unit* and the elements of a time unit are denoted coordinates and represented by integers. The definition of a time unit involves two functions, one mapping a coordinate onto a half open interval on the time line, and a second mapping a point in the time line to a coordinate. Time unit granularity is represented by a function *U_within_V(v,i)* which returns the i-th coordinate of the U time unit that is within the interval of the coordinate v of the V time unit. For example Month_within_Year(2000, 2) would return the month February of year 2000. Ohlbach also defines the notion of sets of coordinates. This extensive functional language allows representing arbitrary calendar systems, and facilitates conversions between them, and also allows expressing notions such as weekend, office hours etc. Since this representation is based on integers, the advantage of symbolic representation of time is lost.

Ning, Wang, and Jajodia [12] propose an algebraic approach to defining calendars, which is an instantiation of the general framework proposed in [1]. A calendar is expressed as a collection of granularities which are derived through algebraic operations from a single 'bottom' granularity. For example, if 'day' is chosen as the bottom granularity, then 'week' is derived by applying the grouping operation. Other interesting operations they propose are the altering-tick operation that allows to shrink or expand a granule (allows to represent months with different numbers of days) , the shifting operation that allows to define a new granule based on the shifting of an existing granule (good for representing time zones), and other set based operations. This scheme allows the computation of granule conversions, which basically lets one answer questions like "What is the date of the first Monday of September 2001". The authors contend that this representation is natural, and allows for compact representations. We find that the operations require deep understanding, and it would be difficult for a casual user to define a calendar system using this representation.

Snodgrass et al. have been concerned with temporal representation issues in the domain of RDBMs [9][16][17]. They point out that many different calendars exist, and that the value of a particular calendar is determined by the population that uses it. The authors discuss the inconvenience of not having the ability to use calendar expressions in DB applications, and propose an approach for supporting internationalized time constants in the context of a DBMS. They propose an augmentation of SQL with three new data types: event, that denotes a time point, interval, that denotes a period in time, and span, that denotes a duration. The DBMS system architecture is modifies to support extensibility, the users can define new calendars and calendar systems. These calendars can be used in the context of SQL statements and declarations.

6 Conclusion

We have developed and demonstrated an object oriented model and framework for the definition of arbitrary periodic calendars. This framework can be added to Calendaring and Scheduling applications to facilitate the customization of such systems for specific domains and organizations. The model deals with intuitive temporal concepts such as time units, time cycles, and dates, and presents them in an intuitive manner, allowing a user to quickly master the framework and easily define a calendaring system.

References

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

[2] F. Dawson, D. Stenerson, "Internet Calendaring and Scheduling Core Object Specification" (iCalendar) Network Working Group RFC: 2445 November 1998
http://www.faqs.org/rfcs/rfc2445.html

[3] Dyreson, C. et al. Efficiently supporting temporal granularities. Time Center, Technical Report, TR-31, 1998.

[4] C. Dyreson, R. Snodgrass, and M. Freiman. Efficiently supporting temporal granularities in a DBMS. FTP, Technical Report 95/7, James Cook University, Australia, 1995.

[5] Greg FitzPatrick, Calendaring and Scheduling with XML-RDF, XML Conference & Exposition 2001, Orlando, Florida, December 9-14, 2001.

[6] Goralwalla, I. et al. Modeling temporal primitives: Back to basics. In Proceedings of the 6th International Conference on Information and Knowledge Management (CIKM-97), pages 24-31, 1997.

[7] Goralwalla, I. et al. Temporal granularity: completing the puzzle. Journal of Intelligent Information Systems, 16(1):41-63, 2001.

[8] Internet Mail Consortium, vCalendar - The Electronic Calendaring and Scheduling Exchange Format, http://www.imc.org/pdi/vcal-10.txt, September 18, 1996.

[9] N. Kline, J. Li, and R. Snodgrass. Specifying multiple calendars, calendric systems, and field tables and functions in TimeADT. Time Center, Technical Report, TR-41, 1999.

[10] S. Kraus, Y. Sagiv, and V. Subrahmanian. Representing and integrating multiple calendars. University of Maryland, Technical Report, CS-TR-3751, 1996.

[11] H. Lin. Efficient conversion between temporal granularities. Time Center, Technical Report, TR-19, 1997.

[12] P. Ning, X. Wang, and S. Jajodia. An algebraic representation of calendars. *Annuals of Mathematics and Artificial Intelligence*, 36(1): 5-38; Sep 2002.

[13] H. Ohlbach and D. Gabbay. Calendar logic. *Journal of Applied Non-classical Logics*, 8(4):291-324, 1998.

[14] H. Ohlbach. About real time, calendar systems and temporal notions. In H. Barringer, et al., editors, Advances in Temporal Logic, Kluwer Academic Publishers, pages 319-338, 1999.

[15] E. Reingold and N. Dershowitz. Calendrical calculations: The millennium edition. Cambridge University Press, 2001.

[16] Richard Snodgrass, and Michael Soo, The MultiCal Project - Overview of MultiCal. In http://www.eecs.wsu.edu/~cdyreson/pub/temporal/papers/multical.pdf, November 1993.

[17] M. Soo and R. Snodgrass. Mixed calendar query language support for temporal constants. TempIS, Technical Report 29. Computer Science Department, University of Arizona, 1992.

[18] X. Wang. Algebraic query languages on temporal databases with multiple time granularities. In Proceedings of the International Conference on Information and Knowledge Management (CIKM), Baltimore, Maryland, pages 304-311, 1995.

[19] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with granularity of time in temporal databases. In Lecture Notes in Computer Science, vol. 498, R. Anderson et al., editors, Springer-Verlag, 1991.

[20] R. Zhang and E. Unger. Calendar algebra. Kansas State University, Technical Report, 1996.