An OASIS WS-Calendar White Paper

Conceptual Overview of WS-Calendar WD01

Understanding inheritance using the semantic elements of web services

By Toby Considine
On behalf of the OASIS WS-Calendar Technical Committee

Date: 11 September 2010

OASIS

WS-Calendar defines calls and semantics to perform temporal alignment in web services interactions. Short running services traditionally have been handled as if they were instantaneous, and have used just-in-time requests for scheduling. Longer running processes, including physical processes, may require significant lead times. When multiple long-running services participate in the same business process, it may be more important to negotiate a common completion time than a common start time. WS-Calendar extends the well-known semantics and interactions built around iCalendar and applies them to service coordination. This white paper explains some of the issues in generic service coordination as an aid to understanding how and when to use WS-Calendar

This white paper was produced and approved by the OASIS WS-Calendar Technical Committee as a Committee Draft. It has not been reviewed and/or approved by the OASIS membership at-large.

Table of Contents

Why WS-Calendar, why now?

As physical resources become scarcer, it is imperative to manage the systems that manage our physical world just as we manage business and personal services. The controlling paradigm of our resources shifts from static efficiency to just-in-time provision of services. At the same time, technology and policy are moving toward reliance on resources that are intermittently available, creating another constantly changing schedule. The challenge of the internet of things is to manage the collision of these schedules.

Service oriented architecture has seen growing use in IT as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It is natural to think of one computer agent's requirements being met by a computer agent belonging to a different owner. The granularity of needs and capabilities vary from fundamental to complex, and any given need may require the combining of numerous capabilities while any single capability may address more than one need. SOA is seen to provide a powerful framework for matching needs and capabilities and for combining capabilities to address those needs. The purpose of using a capability is to realize one or more real world effects. When we expose these capabilities for remote interaction, we refer to it as a service.

Physical processes are already being coordinated by web services. Building systems and industrial processes are operated using oBIX, BACnet/WS, LON-WS, OPC XML, and a number of proprietary specifications including TAC-WS, Gridlogix EnNet, and MODBUS.NET. In particular, if building systems coordinate with the schedules of the building's occupants, they can reduce energy use while improving performance.

Service interactions have typically lacked a notion of schedule or of temporal coordination. Short running services have been handled as if they were instantaneous, and schedules have been managed through just-in-time requests. Longer running processes, including physical processes, may require significant lead times. Long-running processes have different dynamics than do short ones. For example, it may it may be more important in some scenarios to negotiate a common completion time than a common start time.

Physical services rely on a diverse mix of technologies that may be in place for decades. Direct control of diverse technologies requires in-depth knowledge of each technology. Approaches that rely on direct control of services by a central system increase integration costs and reduce interoperability. Interaction patterns that increase schedule autonomy free up such systems for technical innovations by reducing the need for a central agent to know and manage multiple lead times.

An increasing number of efforts are underway that require synchronization of processes on an "internet scale". Efforts to build an intelligent power grid (or smart grid) rely on coordinating processes in homes, offices, and industry with projected and actual power availability; these efforts envision communicating different price schedules at different times. Emergency management coordinators wish to inform geographic regions of future events, such as a projected tornado touchdown. The open Building Information Exchange specification (OBIX) lacks a common schedule communications for interaction with enterprise activities. These and other efforts benefit from a common cross-domain, cross specification standard for communicating schedule and interval.

WS-Calendar builds on iCalendar

For human interactions and human scheduling, the well-known iCalendar format addresses these problems. Prior to WS-Calendar, there has been no comparable standard for web services. As an increasing number of physical processes become managed by web services, the lack of a similar standard for scheduling and coordination of services becomes critical.

WS-Calendar is part of a concerted effort to address the issues above. CalConnect, working through the IETF, has updated the RFC for iCalendar to support extensibility [RFC 5545]. They have submitted a standard for XML serialization of iCalendar which the WS-Calendar specification relies on heavily.

The intent of the WS-Calendar technical committee was to adapt the existing specifications for calendaring and apply them to develop a standard for how schedule and event information is passed between and within services. The standard adopts the semantics and vocabulary of iCalendar for application to the completion of web service contracts. WS Calendar builds on work done and ongoing in The Calendaring and Scheduling Consortium (CalConnect), which works to increase interoperation between calendaring systems.

Building on iCalendar's Components

The iCalendar object includes many elements to support distributed scheduling and authorization for events. Transactions are committed based upon distributed decisions communicated by systems that are frequently off-line. Calendar management is a rich and complex problem whose solutions and techniques are robust and mature. WS-Calendar includes service definitions to invoke these behaviors.

At the heart of the iCalendar message is the components collection. WS-Calendar extends the semantics of these components to meet the needs of service integration.

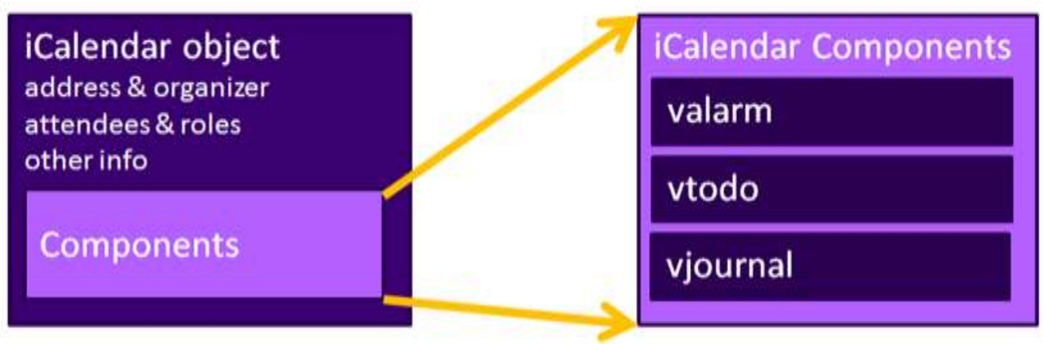

Figure 1: iCalendar specifies scheduling components that are well known and well understood

Don't worry. You won't see the iCalendar components (vobjects) again. WS-Calendar inherits behaviors and attributes form the iCalendar components to define the Interval, the Sequence and the Calendar Gluon. WS-Calendar builds services scheduling and performance alignment upon these three components. Because of the inheritance, these objects can travel within and interact with the world of today's calendars.

Semantic Components of WS-Calendar

WS-Calendar semantics define a structure for the common expression of schedules for events or a series of events. Because physical processes may require other supporting services, scheduling of the services described in these structures may be constrained in performance; you can't schedule a reception at a hotel without also scheduling a set-up and a clean-up. WS-Calendar enables the expression of such relationships without requiring the calling party to understand the supporting processes.

Other processes may involve parameterized negotiations between services. Intervals may be of fixed or variable duration. Purchase prices and quantities may vary over time. The intervals may be consecutive, or intermittent. WS-Calendar provides a common mechanism for elaborating these details using inheritance and local over-rides to enable remote invocation, controlled patterns for service specification, and two-way negotiation while achieving parsimonious serialization.

The Core Components

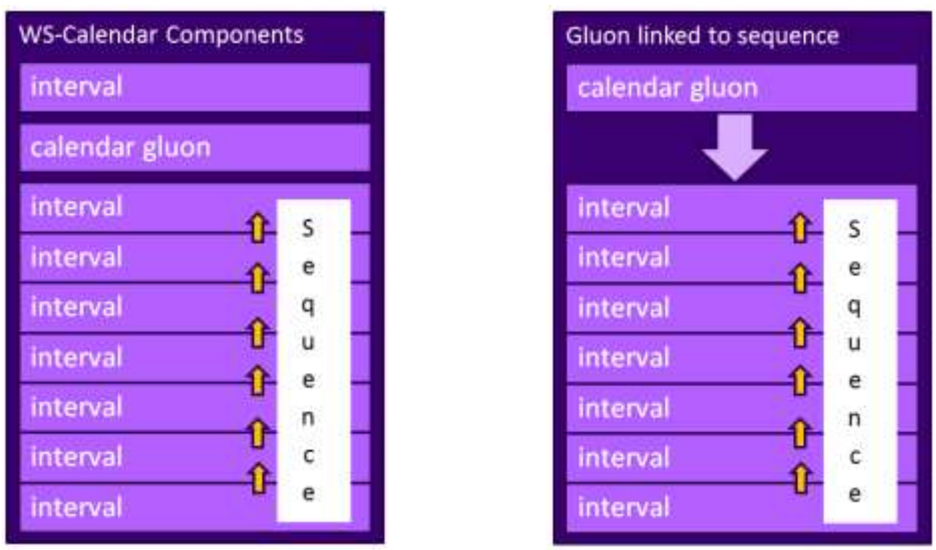

Figure 2: Intervals and Calendar Gluons

The core components of WS-Calendar are the Interval and the Calendar Gluon. Each of these inherits definitions and structure from the iCalendar components.

Intervals

The Interval is a length of time associated with service performance. Each interval has a defined payload of XML information. When an interval has a scheduled start time or end time, then we call it a Scheduled Interval.

iCalendar components include Relations, whereby the message publisher can specify relationships between components. The iCalendar relationships are PARENT, CHILD, SIBLING, START, and END. WS-Calendar extends this list with Temporal Relationships: STARTFINISH, STARTSTART, FINISHSTRT, FINISHFINISH, each with an offset expressed as a duration. Intervals and relationships together define Sequences.

Sequences

A Sequence is a collection of intervals with defined temporal relationships. The simplest sequence is set of consecutive intervals of the same duration. WS-Calendar names such a simple, regular Sequence a Partition.

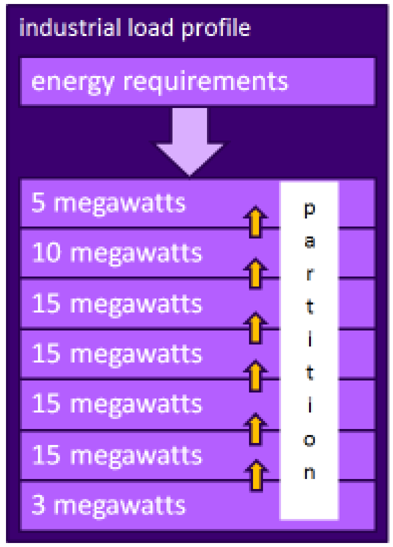

Figure 3: The Partition, the simplest Sequence

Figure 3 depicts a simple repeating time interval along with a single external expression of the type of information provided by each interval. In Figure 3, it is labeled Energy Requirements; in WS-Calendar, this is an instance of a Calendar Gluon (see below).

The intervals in a sequence have a coherent set of relationships between them. The collection of Intervals in Figure 3 defines a period of time, but not a particular period; there is no start or end time for any of the Intervals. If a single interval of a Sequence is scheduled, one can compute the schedule for each of them. A particular service interaction can schedule the Sequence by defining a Start Date and Time. Another interaction could schedule the same Sequence again with a different Start Date and Time.

Calendar Gluons

Calendar Gluons hold information to define an interval. Any information specified in an Interval can also be specified in the Calendar Gluon. So why have a Calendar Gluon?

In physics, Gluons act to mediate as well as to participate in the interactions between quarks. A Calendar Gluon defines information to be inherited by each Interval in the Sequence, as well as scheduling the entire sequence.

Referring again to the Industrial Load Profile in Figure 3, the Calendar Gluon specifies that each Interval is defining Energy Requirements. The amount required varies by each interval, but the service of each Interval is the same. Collections of such similar intervals are useful in energy and other markets involving volatile resources.

Repeating intervals are interesting in day-to-day interactions because they are the way many services already are delivered. It is useful to be able to vary a Sequence parametrically. Take, for example, classroom scheduling at a College. It is typical for class schedules to use one hour intervals on Monday, Wednesday, and Friday. Classes scheduled on Tuesdays and Thursdays are of 50% longer duration to establish an equivalent in classroom time for classes taught on the two schedules.

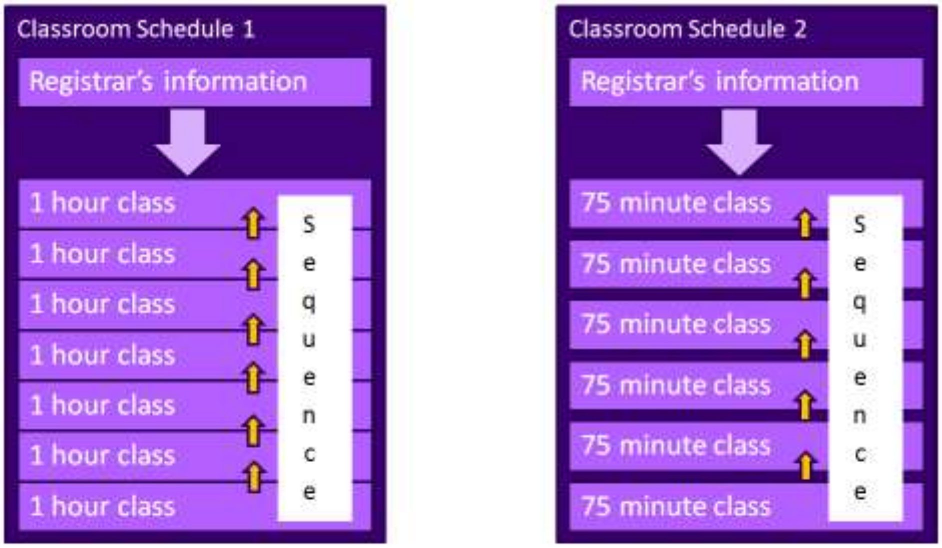

Figure 4: Classroom Schedules

Classroom Schedule 1 shows a schedule for one hour classes. Classroom Schedule 2 illustrates an every hour and a half schedule for classes, with 15 minute breaks built in.

The duration of each Interval, and the relationship between each interval and the preceding one, can be expressed within each interval using the Temporal Relationships. For a regular sequence such as those in Figure 4, it is much simpler to express the duration and relationship once, in the Calendar Gluon. All Intervals in the Sequence will inherit those elements unless overridden.

Summary

WS-Calendar uses the Interval, the Sequence, and the Calendar Gluon to define repeating instances of service performance. Inheritance within Sequences allows parsimonious serialization as well as specific use for a variety of purposes.

Assembling Business Objects using WS-Calendar

This section provides an overview of how to build regularly recurring temporal service structures using inheritance. It also discusses how to override that inheritance when you need to.

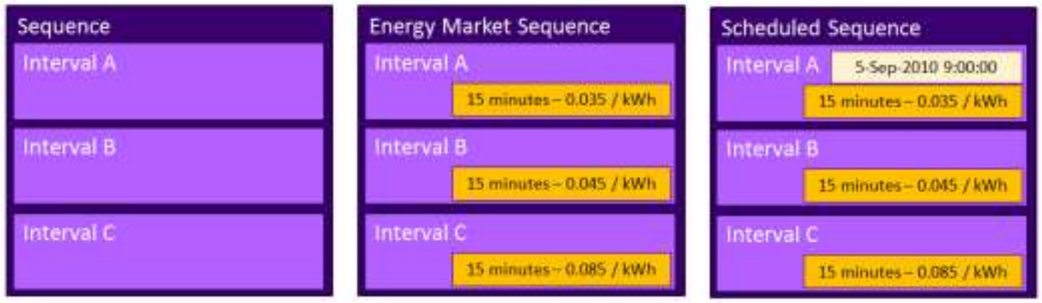

Figure 5: Building a Sequence into a Business Service

In Figure 5, we start with a simple Sequence. To each interval, we can add some contract or service information. Finally, we schedule the Sequence by adding a single start date to the whole Sequence.

Inheritance

Figure 6: Inheriting Duration from an Calendar Gluon

We can reduce the amount of repetition using a Calendar Gluon to create a default duration for the Sequence. In Figure 6, Sequence 1 and Sequence 2 are identical.

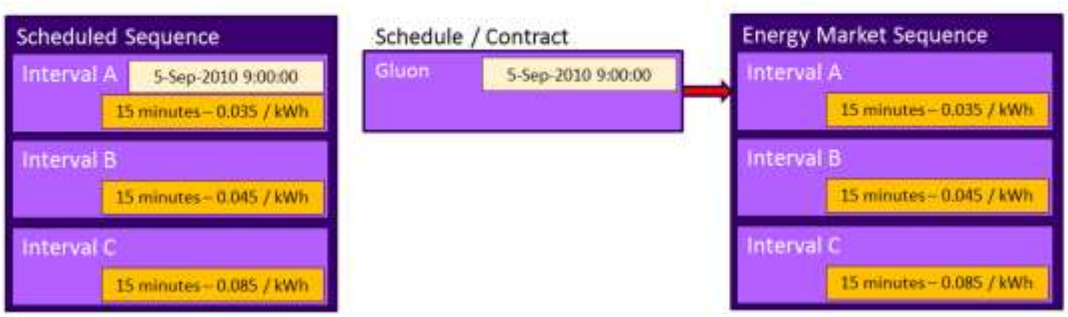

Figure 7: Inheriting Schedule from an Calendar Gluon

In a similar way, Figure 7 shows two identical Sequences, one inheriting a schedule from an Calendar Gluon that indicates that Interval A starts at a particular date and time. Note that inheritance of a Scheduling option is unique in that it sets the time only on the Interval mentioned in the Relationship. This is because all Intervals in a Sequence become scheduled when any member of the Sequence is scheduled.

Stacking Inheritance

Calendar Gluons can also be related recursively, that is, WS-Calendar supports defining a Calendar Gluon with another Calendar Gluon, and thereby with the entire sequence.

Figure 8: Introducing Stacked Inheritance

In Figure 8, the Sequence is scheduled by adding a Calendar Gluon to an existing Calendar Gluon. The pre-existing Calendar Gluon defined the service offering and the default interval (15 minutes) for the Energy Market Sequence. The pre-existing Calendar Gluon also defined Interval A as the entry point for the sequence, i.e., any schedule established will be applied to Interval A.

This use of a Calendar Gluon enables some interesting service behaviors. A Sequence and a Calendar Gluon can define a complete service, with the entry point defined by the Gluon. We might call this service a market Offering. Another party can contract that offering by referencing the existing intact Sequence as referred to by the Calendar Gluon. In market service interactions, scheduling a service is calling for execution of a contract. Stacked inheritance enables a clean separation of product definition and market call for execution.

Figure 9: Second Stacking Inheritance example

Figure 9 shows a different use of stacked inheritance. A catering system interacts with a standard contract for the HVAC system to support a reception in a hotel. The building system integrator created standard contracts in the Energy Management System (EMS) for those activities that are invariant. The standard contract leaves indeterminate those elements that vary for each catering job. The Sequence is assigned a name and an entry point using the Calendar Gluon.

At a later time, the catering software invokes this defined offering, associating the schedule and the capacity requirements to make a contract. Through inheritance, only the “Event” interval is changed, receiving a capacity (to influence ventilation) and the duration for the reception. Because the exposed Calendar Gluon indicates that the “Event” is the entry point, the reception schedule for 9:00 schedules the series so that the “Event” begins at 9:00. The catering software requires no knowledge of the support services offered in other intervals.

Once the contract is created, the room would show up as Busy during room set-up and break-down when standards-based calendaring inquiries are made against the EMS.

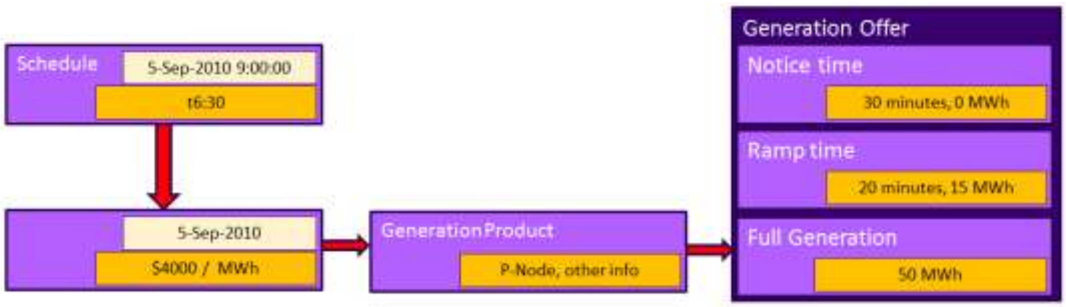

Figure 10: Stacking Calendar Gluons three deep

In the very similar scenario in Figure 10, an energy generation resource has market offering that requires 50 minutes of pre-notification. On September 4th, the generation resource is bid into the next day’s market with a price it is willing to accept. The market operator, schedules energy production and notifies the resource that its bid has been accepted and that its services will be required for six and a half hours.[1]

[1] Note: This is meant to be neither a depiction of today’s markets, nor a recommendation for tomorrow’s. It is merely an illustration of the capabilities and approach.

Advanced Scheduling

The examples so far have included only simple partitions and single schedules. This section illustrates some of the flexibility of the WS-Calendar scheduling model

Multiple Relationships

Key interactions in smart energy involve mutually unintelligible systems coordinating their behavior for the optimum economic result. Today's interactions are machine to machine interactions; tomorrows will be business to business.

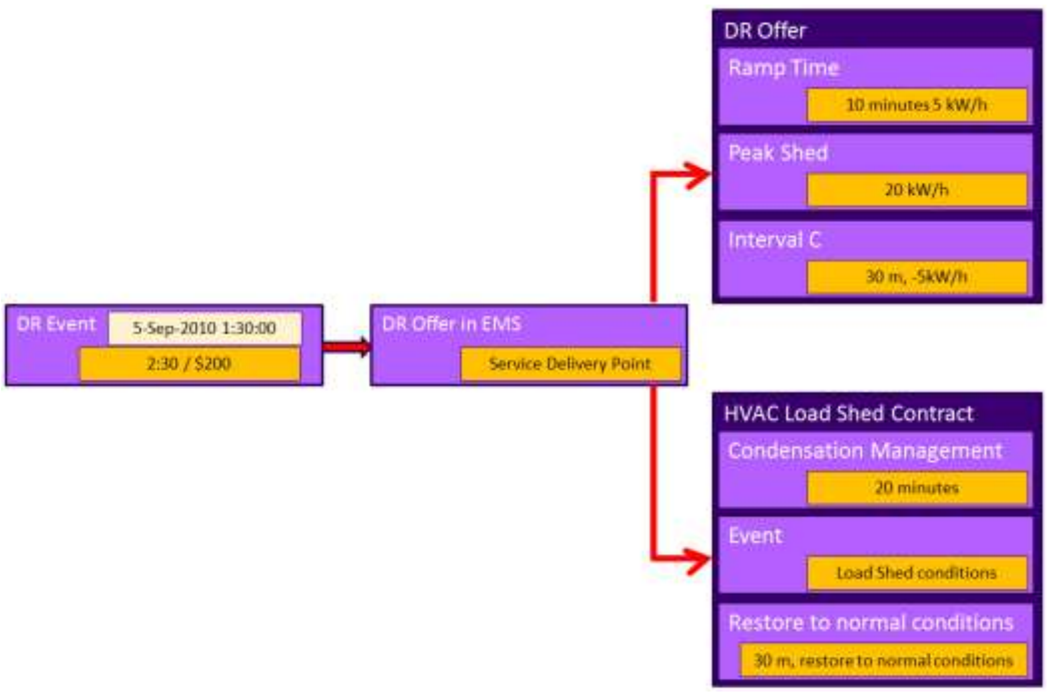

Figure 11: One Calendar Gluon, Two Sequences

Figure 11 illustrates an Energy Management System (EMS), which is offering demand response (DR) to the grid-based markets. The building system integrator has defined the Sequence to shut down certain systems, and then to restore them to full operation afterwards. This is the HVAC Load Shed Contract.

The energy use effect of these decisions appears in a parallel Sequence, herein the DR Offer. Notice that the lead time in HVAC operation is longer than the lead time in DR—the first activities of the HVAC system do not yet reduce energy use. Notice as well, that during system restoration, the building will use more energy than it does during normal operations, indicated by a -5kWh Demand Response.

When the energy supplier sends a notification of a DR Event, it schedules that event to begin at 1:30 and to last for two and a half hours. This offer also comes with a monetary value. When the EMS accepts the offer, it shares the DR event as scheduled with the purchaser, and notifies the building systems of the three intervals in the HVAC contract as scheduled.

Neither the EMS system nor the DR purchaser needs to have any understanding of the underlying systems. Each needs merely to read the WS-Calendar based service attributes.

Classroom Scheduling Revisited

We started this document with an illustration of classroom schedules rendered in WS-Calendar. We now revisit this illustration using the concepts including inheritance and contracts that that paper has illustrated. We started this discussion of Sequences with an illustration of classroom scheduling in Figure 4.

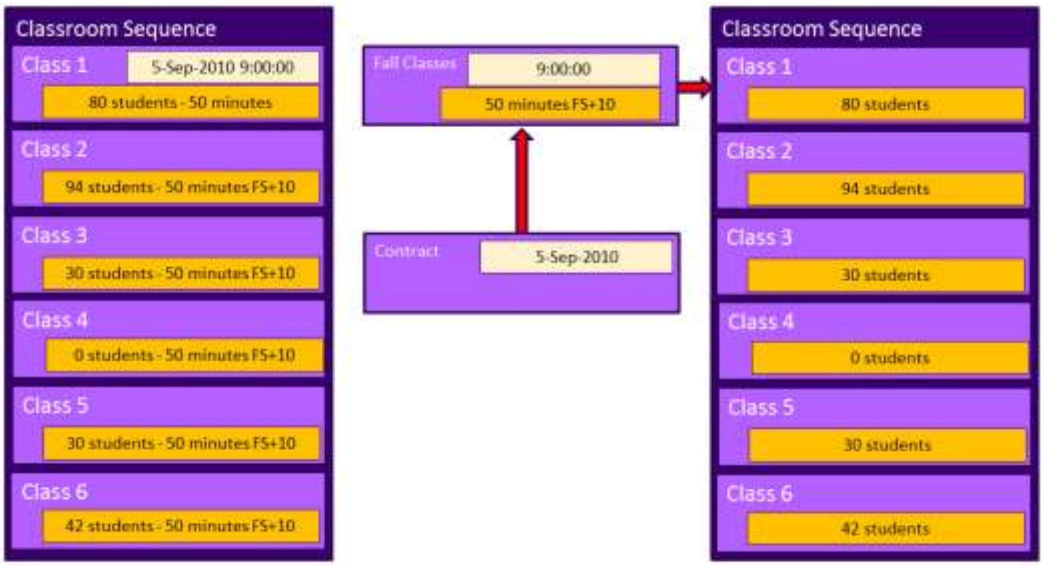

Figure 12: Classroom Schedules Revisited

In Figure 12, we revisit this using the inheritance. In this high-tech classroom, there are systems to warm up, and ventilation levels to be maintained to support each class. The registrar's office puts out a schedule for each classroom indicating how many students will be in it for each of six periods during the day.

The classes are not really an hour long, but are 50 minutes long with a 10 minute break between classes. A Campus EMS creates a schedule with an Calendar Gluon that includes a 50 minute duration and a FINISHSTART relationship with a duration of 10 minutes. Each day begins at 9:00. This is the standard building system contract for Fall Classes.

To finally schedule contract performance, a Calendar Gluon referencing the Fall Classes and the date for each school day during the semester is created.