Applying an Intelligent Reconfigurable Scheduling System to Large-Scale Production Scheduling

Annaka Kalton

Stottler Henke Associates, Inc.
951 Mariner's Island Blvd, Suite 360
San Mateo, CA 94404
kalton@stottlerhenke.com

Abstract

Despite the invaluable role played by scheduling software in a number of domains, the cost and expertise involved in creating a system suited to each new area has restricted the adoption of such tools. To make scheduling software attainable by a broader audience, it must be possible to create new scheduling systems quickly and easily. What is needed is a framework which takes advantage of the large degree of commonality among the scheduling processes required by different domains, while still successfully expressing their significant differences. We briefly describe a framework which distills the various operations involved in most scheduling problems into reconfigurable modules which can be exchanged, substituted, adapted, and extended to accommodate new domains. We then discuss how this system was successfully applied to the large-scale production scheduling involved in airplane assembly.

Introduction

Planning and scheduling software has the potential to benefit a variety of domains and industries. Unfortunately, although there are a variety of high-quality customized scheduling systems available, off-the-shelf systems rarely fulfill the scheduling needs of any one domain. This is, in large part, because domain knowledge is crucial to taming the intractable nature of scheduling problems in general.

The result of this is that the main domains that can take advantage of scheduling systems are either those that can afford a full custom solution, or those that fall within the narrow commercial off-the-shelf domain coverage (e.g. for project planning). Even in the latter case, however, the solution is often a poor fit, because the modeling tools are often limited in their expressiveness, and the scheduling process itself is generic.

This problem is all the more frustrating because many scheduling systems share a variety of common functionality. We believe that the solution to this problem is to create a standard scheduling framework, with parts of the scheduling process broken out into discrete components that can easily be replaced and interchanged for new domains. We have created such a framework in Aurora, a configurable scheduling engine, and successfully applied it to a number disparate domains, including orbiter preparation scheduling and missile intercept assignment. The domain discussed here - airplane assembly scheduling - has a large number of complex resource requirements, temporal constraints, and timing restrictions.

We will begin by giving an overview of the reconfigurable system and its components; we will then discuss how the system was applied to the problem of airplane assembly scheduling; finally, we will discuss our conclusions and possible future directions for this research.

Reconfigurable Scheduling Framework

Aurora was designed to be a highly flexible and easily customizable intelligent scheduling system. To achieve this goal, we designed it to have a number of components that could be plugged in and matched to gain varied results.

The scheduling system permits arbitrary flexibility by allowing a developer to specify what code libraries to use for different parts of scheduling. Each of the pluggable components must extend the corresponding general base class that defines the entry-point methods. This allows the objects that are integral to Aurora to interact with them successfully. The libraries may make use of any of the Aurora objects (such as activities and resources) that pass through the interface. These objects provide support for additional attribute caching, permitting domains to make use of custom properties in the scheduling heuristics.

The primary pluggable components include a preprocessor; a scheduling queue prioritizer; the actual scheduler, which usually applies several scheduling methods; a conflict solution manager; and a postprocessor. See Figure 1 for a more detailed breakdown of configurable operations.

Some of the pluggable components are independent elements; others may depend on the existence of parallel schedulable components in other parts of the scheduling process.

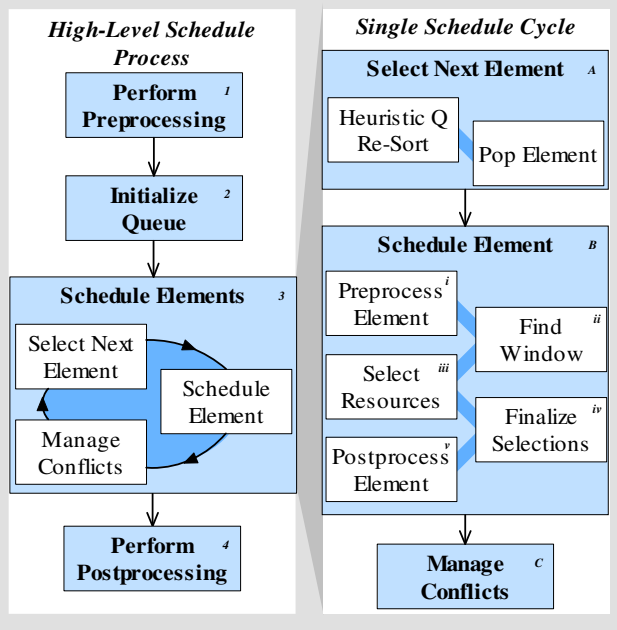

Figure 1. Aurora's reconfigurable scheduling system process breakdown. This shows a few of the configurable decision points, and will be used for scheduling stage cross-reference in the customization discussion.

Airplane Assembly Scheduling Domain

Extremely large-scale production scheduling, such as airplane assembly scheduling, is a challenging real-world scheduling problem that offers a number of interesting features and considerations, some of which are discussed below. We successfully reconfigured Aurora to satisfy these special requirements without violating the general scheduling solution framework; this customization is discussed in the following section.

Extensive temporal constraints. The assembly plan for a single airplane may have over two thousand jobs, and over six thousand temporal constraints. Although the plane is assembled in several sections, forming several especially heavily constrained sub-networks, there are also a significant number of constraints among these sections. This would not be problematic except that they combine with extensive resource requirements to form an extremely heavily constrained problem definition.

Extensive resource requirements. Almost every job in the assembly plan has at least one resource requirement; and many have more than twenty. The required resources often function at or near capacity.

Variable resource capacities. The personnel resources and some equipment resources have a capacity that varies over time. This variability is modeled as a nested variance description: the patterned variability within a given time period (e.g. 35 mechanics for 8 hours, 20 mechanics for 8 hours, and 7 mechanics for 8 hours; repeat); and different patterns across different time periods (e.g. manpower across shifts may be different in April than in March).

Workspace consumption. Many of the resources constraining the schedule are in fact work zones, reflecting the fact that most tasks must be performed in a specific location, and only so many people can be at that location at a time. However, in the course of airplane assembly, most of these zones are effectively eliminated by jobs that install hardware which prevents subsequent access to the area. Such work zones are not true consumable resources; in most cases the resource's client job does not diminish the resource's capacity beyond that job's duration. A zone might be required by 100 jobs, the last two of which eliminate it.

Interacting calendars. Rather than having a single work calendar - either globally or at a job level - there are a number of calendars that must interact dynamically in the scheduling process, with the scheduler taking the intersection of all applicable calendars to find a correct result. An example of this would be a plane's work calendar being combined with a job's and multiple resource calendars to find the actual workable windows.

Soft scheduling. Some jobs may be split into multiple jobs if it improves the schedule; other jobs may occur at the same time as certain compatible jobs, even though usually this would produce a conflict and invalid schedule. These soft scheduling attributes help produce a shorter schedule that is still workable and acceptable, but also add a degree of challenge to finding that schedule quickly.

Analysis complexity. The scheduling problem itself is rich and complicated. However, it is not sufficient for the system to produce a feasible schedule; it must also produce a comprehensible schedule. The scheduling team is continually trying to improve the production plan's formulation to gain a plan that can be scheduled more compactly. In order to do this, the scheduling team must not only be able to extract a feasible schedule from the system: they must also be able to look at the schedule and gain an understanding of why it scheduled the way it did, so that they can focus on those parts of the production plan that could result in schedule cycle improvement if streamlined.

Airplane Assembly Scheduling Customization

The greatest consideration in airplane assembly scheduling is the scale of the problem, and the tangle of inter-related constraints which make it extremely difficult to fix conflicts. These are the fundamental driving factors for this domain; other considerations add to the domain's complexity, but in and of themselves do not make the scheduling difficult. Below we discuss how we addressed each of the airplane assembly scheduling considerations introduced above. Each section finishes with a cross-reference indicating which stages in the scheduling process (see Figure 1 for references) had to be modified to accommodate the change.

Extensive constraints. The resource requirements make conflicts likely, while the temporal constraints make it very difficult to successfully resolve these conflicts. Rather than attempting to fix these conflicts, we instead focused on conflict-free scheduling and the scheduling order that could result in such scheduling. By not trapping earlier activities into restricted windows by scheduling later activities first, we could avoid conflicts; subsequent work focused on prioritization heuristics which tended to result in shorter schedules. For example, scheduling jobs with a large number of down-stream dependencies (not only direct successors, but successors' successors, etc.) tends to result in a schedule with a shorter cycle time. Adding resource requirement considerations to this analysis improves results even more.

Customizations focused on stage 1, for heuristic initialization; and stages 2 and 3.A, for queue prioritization and management.

Variable resource capacities. The challenge with variable capacities lies in modeling them efficiently. In the scheduling process itself, the variable capacities are no different from having a number of activities already scheduled, occupying certain patterns of usage within a resource. We considered modeling the capacities in exactly that way - using "dummy" activities - but concluded that the number of objects required would be prohibitive. Instead, we apply a capacity pattern (made up of nested capacity routines) to the linked lists of time slots that express a resource's availability through time. Because the resource knows what its capacity plan is, and it is easy to find the routine for a given time window, the plan can be applied on an as-needed basis, significantly improving overall performance by minimizing the number of resource time slots required. See Figure 2 for an example of a schedule using variable capacities.

Customizations focused on stage 3.B.iii, for incremental capacity plan application on an as-needed basis.

Workspace consumption. Because most work-zone client jobs do not consume them (the work takes place and the person leaves, freeing the zone for another task), modeling the zones as true consumable resources did not give the desired flexibility. Nor did the application of temporal constraints from jobs to resources to dictate resource end time, because in some cases a job only consumed part of the zone. To accurately reflect the desired flexibility we used a new type of constraint - a consumption constraint - which would remove the associated quantity from the resource's capacity once the job was scheduled. We also augmented the job prioritization to guarantee that all work making use of the zone would be complete before it was fully consumed, preventing unnecessary conflict resolution.

Customizations focused on stage 3.A, for zone-availability priority maintenance; and stage 3.B.v, for consumption constraints propagation to eliminate the associated resource capacity.

Interacting calendars. Intersecting calendars is not a fundamentally difficult problem; the challenge lies in the

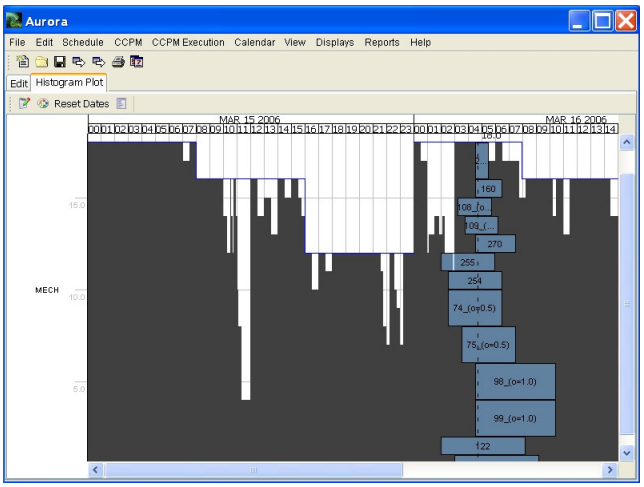

Figure 2. The histogram display in Aurora, showing details for a specific time slice and reflecting a variable per-shift capacity for the mechanics.

sheer number of elements involved. This difficulty was ameliorated by the fact that there are generally fewer than ten calendars in use overall; so rather than intersecting each job/plane/resource set combination, we could cache each calendar combination for later lookup and use. In most cases, one of two or three composite calendars was appropriate, with a few exception cases making use of a less standard composite. The compilation and cross-referencing could then be done on an as-needed basis and cached for future reference; in general each such analysis only had to be done once for a given plan, unless the calendar definitions changed.

Customizations focused on stage 3.B.ii, for calendar retrieval and (if necessary) compilation.

Soft scheduling. The primary challenge of the soft scheduling is the question of whether to apply it. Splitting a job into two pieces, for example, is certainly desirable if it buys you six hours in overall flow time. What if it buys you an hour? Half an hour? Ten minutes? Because it is not always worth the implicit cost necessary to take advantage of the soft scheduling options, we folded the analysis involved into the single-step post-processing (stage 3.B.v): methods called just after a given element is scheduled, before the next is scheduled. This allows the methods to alter the decisions made in the course of scheduling, but also take advantage of the knowledge gained to make more intelligent decisions.

Customizations focused on stage 3.B.v, for considering the utility of softening the schedule selections made in earlier 3.B stages.

Analysis complexity. Providing transparency to complex scheduling decisions, especially in a schedule of such scale, is a great challenge. We focused on the more straightforward question of why a given element scheduled where it did: what temporal constraints impacted the possible time window and how; did resource availability affect the selected window; and if so, on what other job

was this job waiting. Based on this information the system produces a layered explanation reflecting the nested possible time windows: what elements restricted a given window, and why. It also maintains direct links to the elements that determined the final decisions, so that from the GUI the user can easily navigate along a string of inter-related elements to better understand the root cause of a cascading series of scheduling decisions. The GUI also provides a number of drill-down displays and reports, providing a greater degree of transparency to the actual scheduling results. This information allows a savvy user to gain an understanding of both the broad results and specific decisions, providing the tools he needs to manipulate the assembly plan definition to improve the overall schedule cycle. See Figure 2 for an example of a high-level display with drill-down exploration support.

Customization focused on stages 3.B.i, 3.B.iv, and 3.B.v, for caching information about constraint propagation impact and actual scheduling decisions.

Related Work

Previous research considering the design and implementation of reconfigurable scheduling systems has built on concepts initially explored in the area of reusable domain analysis [Ferré and Vegas 99, Arango 94] in order to take advantage the similarities between scheduling problems. Most notably, Carnegie Mellon's Intelligent Coordination and Logistics Laboratory has developed the OZONE (O3, or, Object Oriented Opis) scheduling framework [Smith et al 96 and 97]. OZONE, like Aurora, provides the basis of scheduling solution through a hierarchical model of components to be extended and evolved by end-developers. [Becker 98] describes the validation of the OZONE concept through its application to diverse set of real-world problems, such as transportation logistics and resource constrained project scheduling.

A less flexible but more easily configured system was explored in relation to genetic algorithms [Montana 01]. This approach has the advantage of defining schedule properties using a straightforward set of meta data, with much of the more refined optimization configuration being performed automatically using a genetic algorithm. This results in a very easily reconfigured system, but its strength is also a weakness when faced with a complex domain: it is easily configured within certain bounds, but it cannot easily be more completely tailored for a complex domain. Nor can it easily accommodate significant add-on functionality; it primarily draws on a toolkit of standard techniques.

Conclusions

A reconfigurable intelligent scheduling framework using standard scheduling functions, combined with domain-specific components and information, has the potential to allow quick and easy creation of a scheduler well tailored to a specific domain. The price to this is a reduction in computational efficiency; the exact impact depends on the configuration of the components. We would argue that for most domains, this tradeoff is a reasonable one.

We have shown that in its current form, a reconfigurable scheduling system can successfully be applied to a very complex real-world domain. The extensive requirements of this domain could be handled within the limits set by the current configuration boundaries, while making use of the robust temporal and resource-based scheduling methods which we had developed for previous domains.

Further work needs to be done in adjusting the component boundaries, making them sufficiently discrete for swapping in and out while maintaining sufficient transparency for efficient operation. It is possible that a finer granularity of component configuration could prove valuable. It would also be worthwhile to consider ways of taking such a configured scheduling system, and trimming it down for more efficient operation in a specific domain.

References

Arango, G., 1994. *Software Reusability, Ch 2*. Domain analysis methods, pages 17-49. Workshops M.E. Horwood, London.

Becker, M.A., 1998. "Reconfigurable Architectures for Mixed-Initiative Planning and Scheduling," *PhD Thesis*, Robotics Institute and Graduate School of Industrial Administration, Carnegie Mellon university, Pittsburgh, PA.

Ferré, X. and Vegas, S. 1999. An Evaluation of Domain Analysis Methods. *Proceedingss of 4th International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design.*

Montana, David, 2001. A Reconfigurable Optimizing Scheduler. *Proceedingss of the Genetic and Evolutionary Computation Conference (GECCO).*

Smith, S.F., O. Lassila & M. Becker. 1996. Configurable, Mixed-Initiative Systems for Planning and Scheduling. In: Tate, A. (Ed.). *Advanced Planning Technology*. Menlo Park, CA: AAAI Press.

Smith, S.F. & M. Becker. 1997. An Ontology for Constructing Scheduling Systems. *Working Notes of 1997 AAAI Symposium on Ontological Engineering*. Stanford, CA: AAAI Press.