# Rescheduling in Support of Space Operations

**Steven Jowers**

The Boeing Company
13100 Space Center Blvd.
Houston, TX
Steven.Jowers@boeing.com

**Abstract**

This paper provides a brief overview of the rescheduling problem from the perspective of an interval-based, non-chronological scheduler followed by a discussion of how we have solved the rescheduling problem. It will conclude with examples taken from complex aircraft assembly line operations, from a complex, resource rich test facility, and a discussion of how significant aspects of these problems are similar to ones encountered in space operations.

## Introduction

Often our attention is focused on crafting an initial schedule, but once that schedule goes into actual operation, a different, but just as large a problem comes into existence - that of maintaining the schedule as it undergoes the stresses and strains of real-time operations. Besides being completed, tasks are delayed, only partially completed, deleted, and split into one or more new tasks as each day passes. This can quickly render the original schedule obsolete. Rescheduling for continued operations must consider how to manage these perturbations. Can we rebuild the entire schedule from a specified day forward, or can we perturb only parts of the schedule? Is it worth the time and effort to re-optimize? How can we track whether or not the perturbations are causing us to slip schedule? Has the set of critical resources changed? Should additional resources, if possible, be made available? Finally, any solution answering these questions must scale to very large, complex schedules.

## The Rescheduling Problem

The problem domains needing a scheduler to support operations are quite varied. Test facilities, small payload operations, and simple on-orbit maintenance activities can all call out tasks with resource requirements and few, if any, predecessor/successor (precedence) constraints. Other problems, such as payload operations, complex maintenance operations, and on-orbit assembly operations will all have relationships between tasks such as precedence, relational (task to task), and temporal (task to time point) in addition to complex resource requirements.

Support for space operations necessitates scheduling of ground facilities (training and operations), mission planning prior to flight, and mission operations during flight. For space systems with long duration missions (e.g., the international space station), the distinction between mission planning prior to flight and mission operations during flight is quite blurred; once the mission starts there is no a priori "mission planning" prior to flight for everything anymore. Instead, the vehicle is in continuous mission operations. In this latter case, a capability to update an already executing schedule to reflect work actually accomplished and real-world, unplanned opportunities with possibly new constraints, and capability to insert significant new pieces into an already executing schedule (plans for new experiments, maintenance activities, etc.) is a must.

When generating the original schedule prior to actual operations, it must be guaranteed feasible with respect to all documented plan input data. Once in execution, however, real-world demands must allow for the selective relaxation/enforcement of constraints; exploration of hypothetical "what if" scenarios, oversubscription of resource constraints normally kept a conservative level but having additional reserve for emergencies, etc.

Once a schedule begins execution, tasks are hopefully performed according to schedule. Experience shows, however, that external factors cause the delay of some tasks (e.g., unexpected lack of power), the execution of tasks out of sequence (e.g., unexpected opportunities, delays incurred by predecessor tasks, etc.). Additionally, tasks may only be partially complete at reschedule time and their durations may have deviated from planned values.

The challenge presented to a scheduling engine builder whose product must support scheduling prior to and during operations is multifaceted. For our work, we have chosen to approach the problem from three vantages:

- development of a rich and extensible vocabulary for modeling task constraints, including task status data
- development of a scheduling engine for the data model
- and the development of controls for the scheduling process that allows users to execute their own heuristics

### Modeling Vocabulary

It is not the intent of this paper to describe all the

constraints that our experience dictates should be in a scheduler for complex problems. Nor is it the intent to justify the statement that the easy addition of new constraints to the vocabulary is highly desirable. However, by providing a partial listing of constraints we believe need support, we can useful provide backdrop for later discussions. The constraint vocabulary includes:

- precedence constraints
- duration
- relational and temporal constraints
- numerous earliest start times/latest completion times
- resource expressions (arbitrarily deep nested ANDed and ORed resource requirements)
- resource reservation/provision, consumption/production
- multiple priorities
- preferences
- interruptability
- exclusive reservation of a resource to a group of tasks; i.e., resource lockout.
- task group membership where a group impacts resource and timeline availability for non-group members

These are just some of the constraints that the vocabulary should support. In addition to these, each task must allow for documenting the task's status. This is critical to the rescheduling process. If not provided as an element of the vocabulary, it is meaningless to talk of tracking a task's status and executing re-scheduling operations to account for that status.

### Scheduling Engine Design Considerations

The scheduling engine builders face a number of hart design issues. They should design a scheduling engine which, at its core, doesn't need to do anything different for tasks having or not having status information. This simplifies complexity of the core engine, helps preserve performance since the same routines are used for all scheduling functions regardless of status information, and relegates the need for understanding the concepts associated with task status to high level data abstractions. It also helps to minimize the number of assumptions coded into the core engine.

Further, builders need to design algorithms such that operations which are not linear have their impact mitigated. Ideally, the time to (re) schedule should increase linearly with data set size.

The time to (re) schedule should be sufficiently fast to support operational considerations; i.e., it is impossible to use a solution for daily operations when the engine needs more than 24 hours of computation time to generate a schedule. For that matter, a significantly faster solution is required to allow exploration of "what-if" scenarios.

The scheduling engine should be capable of handling very large data sets. Scheduling in support of space operations for programs such as the International Space Station will require the scheduler to process many, many tasks against limited resource availability; e.g., maintenance operations, payload operations, on-orbit training, etc.

Finally, the scheduling engine's implementation should support ease of integration with other and perhaps older electronic data systems. This is critically important if there is an existing Information Systems infrastructure that must be utilized to support schedule execution; e.g., systems that log task status.

### Control of the Scheduling Process

The existence of a rich modeling vocabulary and an engine which can process it is only part if the overall solution. Real-world concerns dictate that users must be given controls for the scheduling process itself. Different data sets will exhibit different complexities. The process of scheduling for one data set, which yields good results, may not do so for another data set. Consider the following examples:

1) Provide sufficient control over the scheduling process so those tasks with latest completion times, if possible, are scheduled without constraint violations; i.e., schedule backward from the documented completion times.
2) Support allocation of a specific resource to a defined group of tasks exclusively once that group has been started. Don't allow any tasks outside the predefined group to have access to that specific resource (i.e., setup time can be prohibitive and thereby render some amount of resource idle time more cost effective than securing a higher resource utilization rate).
3) Support the premature release of a resource reserved for a group of tasks even if all tasks in the group are not yet complete.

While far from being an exhaustive list of requirements, this list begins to point to the type of flexibility needed. At a minimum, each constraint in the modeling vocabulary is a candidate a relaxation/enforcement control.

## One Possible Solution

### Modeling Vocabulary

One technology solution, and the approach we have chosen for our work, uses an approach called the "computation of feasible intervals." We have found that this approach to scheduling allows us to model much of the vocabulary mentioned earlier; i.e., we find that we can visualize many of the constraints in the vocabulary as collections of intervals to be intersected with other sets. The re-scheduling problem can be solved quite nicely under this paradigm. It becomes merely the inclusion of a new constraint or two.

### Process Control

To satisfy the need for control over the (re) scheduling

process, provide to users a powerful task and resource selection/sorting scheme. This can be done by providing:

- commands that operate not over the entire set of tasks or resources, but a selected subset.
- operations that use two task lists, the previous selection and current one, to create a new, composite selection; e.g., intersect, add, subtract, etc.
- a rich set of selection criteria or "filters"
- varied sort algorithms which operate over the selected list whose sorting effects can aggregate

Another tool for controlling the process is provided through control of the engine's internal switches or "mode." Such switches control how the engine processes various constraints and the direction of the schedule operation.

Finally, users can invoke scheduling/unscheduling commands at will over the selected data. The commands are sensitive to order, so here is where the effects of sorts will be seen. Our experience has been that these controls are sufficient for controlling a variety of scheduling processes. General, simple command sequences can usually provide good results. More advanced users who know their data can implement their own "heuristics" through a customized series of selection, sort and schedule commands. If a macro-recording tool is then given to them, they can record their commands for selecting, sorting, and scheduling for subsequent use in other sessions.

## Engine Design

There are many, many design considerations that must be examined when building the engine and it is well beyond the scope of this paper to list many of them, much less discuss them. What will be presented are those considerations we believe are particularly relevant to the (re) scheduling problem itself.

**Common Scheduling Function**: One choice of the scheduling technology, computation of feasible intervals, requires that all constraints be ultimately represented as a set of intervals. In building an engine using this approach, it would be wise to design the engine's core so that it has no knowledge about how to "interpret" any type of constraint, but instead only "talks" in terms of lists of time intervals. By making such distinctions, we can design an engine core that is independent of the type of constraint being processed. The addition of new constraints then becomes one where a higher level data abstraction is defined with appropriate routines to translate it into intervals. This makes adding new constraints to the vocabulary a reasonably scoped, understood activity.

The inclusion of status data in support of the (re) scheduling process is one that, under this paradigm, influences the translation of a few other constraints as they are communicated to the engine core. In fact, the *only* field that is affected in our work is the task's "duration" field. One immediate benefit can be seen in the basic algorithm that effects scheduling (placement of unscheduled tasks onto the timeline). The same routine is used to perform both scheduling and re-scheduling. The only difference in re-scheduling is in the modification of a task duration's value if it has a non-default status.

**Constraint Processing**: Tasks have status data that documents state and deltas against original duration values. Scheduling functions always compute an "effective" duration using original and delta information. If no status information has been supplied, the original planned data is also the effective data. If status information has been supplied, effective data will differ from planned data. In either case, the scheduling function is itself unmodified; it needs a duration value and that's what it gets.

Tasks which are completed have an effective remaining duration of zero. Internally, a zero duration task is treated as a milestone with respect to interval calculations. Point solutions have special effects on resource requirements that are reservations -- they are effectively ignored since they cannot affect a resource's availability over an interval of time. Milestones, however, can still effect production and consumption of resources.

Consider a precedence relation between two tasks. If the successor task is completed out of sequence, the engine will "float" it into the future if needed so that the predecessor's constraints are still enforced. The task that floated into the future now is a point solution and therefore its documented resource reservation requirement no longer impacts the schedule.

There are some hidden issues with this approach. First, if the data model has tasks using temporal or relative constraints against the out-of-sequence task it is possible to create a logically inconsistent condition. Secondly, the completion of work out of sequence creates an operational dilemma for successor tasks. The schedule shows successor tasks later in the schedule than the completed but out-of-sequence task. Can these latter tasks be performed? Users can always modify the constraint data to remove these problems, or the engine can be modified to include a more complex behavior. Our experience has been that the current approach is adequate for most needs and the additional complexity required in the engine to effect a more sophisticated behavior warrants a stronger motivation than any we have yet to encounter.

Finally, the tasks data structure can have placeholder field to hold dates of actual task completion if this is important. This data, using the above rationale however, is not used in the scheduling process (though the engine could be modified to use it at the risk of introducing constraint inconsistencies). Should historical data become important, reports can make use of this data.

**Status**: Tasks have one of three values; waiting, inwork, and completed.

*Waiting*: this is a task's default state. The documented task duration will be unmodified when passed to the engine. Should the task be delayed, the waiting status may be augmented with a "fence," another instance of an earliest start time constraint. This is needed to support real-world impacts to the schedule such as the delay of a

task so that other tasks can take advantage of an unexpected window of opportunity.

*Inwork*: Tasks can be continuing or suspended until a user specified date. The suspended feature is needed to support real-world impacts to the schedule. A task which is continuing will be credited with an optionally supplied amount of work that has already been performed and then, during scheduling, placed on the timeline as soon as allowable. If no completed hours are supplied, then the effective duration calculation will not provide any credit.

The optional delta duration is a value to be applied against the documented duration resulting in an effective remaining duration different than the original documented duration. It is used to document a change in the overall duration of the task from the planned value. It can be positive or negative.

Input at the user interface for this data can be quite varied;

- percent complete and, optionally, hours remaining
- hours completed and, optionally, hours remaining
- hours completed since last status operation (if the task is interruptible and worked in many increments) and, optionally, hours remaining.

*Completed*: Task when complete reduce to milestones on the schedule. Optional values can be supplied that, for reporting purposes, can be used to document how long the task actually took. The delta duration value here describes the change from the documented task duration.

Input at the user interface for this data can be varied but is not as complex as the inwork status. Users can supply total hours worked or total hours worked since last status operation. If no hour data is supplied, use the original duration values.

**Interfaces to Legacy Systems**:

All data structures and types have a core set of defined functions, including readers and writers. This includes the status data and all commands. Readers and writers use ASCII and should be designed to facilitate users' inspection of data.

One consequence of this approach is that the data/command can be read from and written to a stream. This allows us to easily build interfaces with other systems. A stream can come from an electronic interface with another system or from an opened file. Specifically, status data from other systems can be output to a file in the format of a scheduling command. This file can subsequently be processed by the scheduling engine.

By initially using a file interface scheme, requirements can be partitioned into information requirements and electronic peer to peer communication requirements. Interfacing activities should first ensure that the required information can be supplied to the engine via a file. Once this supplied data can reliably drive the engine's scheduling behavior in an acceptable and expected manner, attention can focus on automated, live electronic data feeds for gathering schedule status data.

Re-scheduling for an operational schedule can make use of the engine's scheduling mode/configuration to obtain desired behaviors. For example, by selecting tasks which have a firm latest completion date, these tasks and their predecessors can be placed onto the timeline first by scheduling backwards from their latest completion dates. Remaining tasks can then be selected and the mode changed to a forward direction. Tasks will be placed onto the timeline around the previously scheduled tasks.

## The TimePiece Scheduler

The product we have developed that implements the above approach is called *TimePiece*. Below are some examples of how this product and its predecessor, COMPASS, have been used to solve large, complex (re) scheduling problems.

### Aircraft Final Assembly

The schedules controlling the final assembly of a fighter aircraft can be quite large and contain highly inter-related tasks each having a potentially large set of resource constraints. Schedule characteristics for one specific assembly line include:

- 1500+ tasks per aircraft (two distinct precedence diagrams each) with precedence, resource, and relational constraints, group usage, task interruptability, etc..
- Four aircraft in flow at any one time with an expectation to grow to as many as six for a total of well over 9000 tasks represented in the schedule
- (Re) scheduled daily in 54 minutes on a 133 MHz Pentium PC, having 96 MB RAM
- Status data is provided by a legacy, mainframe database to TimePiece via a formatted command file

### Avionics Integration Center

A different problem domain using this same technology is one involving an avionics test facility. Many users submit discrete, unrelated requests for time in the facility. Each request can contain some vary complex resource expressions which can document alternate sets of possibilities, any one of which will satisfy the need. Characteristics of one such lab scheduler include:

- Management of approximately 450 discrete resources
- As many as 250 users to submitting requests into the facility
- Support for multiple aircraft programs all of which contend for the same resources
- Daily processing of new requests against the baseline schedule
- Use of e-mail to notify users of reservation times and equipment allocation

This system uses a COMPASS derivative and schedules the facility in less than 10 minutes when running on a multi-user VAX. Efforts have started to upgrade its design

to use the new product, Timepiece.

Use of this facility scheduling system has helped realize: an order of magnitude jump in test hours scheduled (80 hours per week to 700+ per week), and an order of magnitude reduction in time spent scheduling (from 50+ hours per week down to as low as 2 hours per week. It has been in use since the first quarter of 1995 with dramatic results and cited as an aircraft program "best practice."

### Scheduling for Space

Our products are also in use to support space related activities. Our older product, COMPASS, is in use to by SpaceHAB, Incorporated, to support preliminary scheduling for its missions. NASA/JSC also uses a COMPASS derivative to schedule one if its facilities, the Systems Engineering Simulator. TimePiece, our latest product, uses the same underlying approach to scheduling and implements lessons learned from these scheduling systems.

## Conclusion

At first glance the types of problems encountered in space operations may appear to be sufficiently diverse to require different scheduling approaches. One may be tempted to say that the problem of generating the original schedule and subsequent rescheduling might require different approaches and underlying technology choices. Our experience has been that by understanding the nature of both the scheduling and rescheduling problem, through the choice of a good scheduling technology (calculation of feasible intervals), and through the use of wise design decisions, the same scheduling engine can be used to support a wide range of problem domains, including original schedule generation and its subsequent re-scheduling. What was once a solution requiring a workstation can now be done on a standard PC, and it can be done well.