General Scheduling Service using Intelligent Resource Management Techniques

Dick Stottler[1]
Stottler Henke Associates, Inc., San Mateo, CA., 94404

Abstract

This paper describes a general, automated scheduling solution to handle resource assignment and scheduling challenges for a variety of space mission applications, and how this solution is packaged as a general scheduling service provided in a Service Oriented Architecture (SOA). The full design for the system, the various software methods that are part of that design, and a recently completed prototype are discussed. The techniques described here have been successfully applied to a variety of space mission application and full-scale, operational development of the general scheduling service is proceeding.

I. Problem Description

Managing real-time space mission and space situational related activities is a complex task, especially when considering all of the resources directly and indirectly involved. Space missions and related activities include space-based observations such as commercial overhead imagery, communication link scheduling (both for health monitoring and commanding and for communication users of communication satellites), ground based observations of space objects, launch support, orbit maneuver support, reacting to space weather events and notifications, and military operations. Space-related resources include the space systems themselves such as satellites and their components, and terrestrial systems such as ground stations, and communication links. The resources and activities to be scheduled are satellite payloads and busses, maneuvering the satellite, modifying the communications, processing management, sensor and data collection management, and satellite communication network scheduling. Therefore, there are several resource assignment and optimization issues associated with space missions, requiring a large number of both separate and integrated scheduling and resource optimization applications.

An automated system is clearly called for to handle both the complexity and time constraints. Actions resulting from the responses to simultaneous events may interfere with each other or there may be an opportunity for synergy. Additionally, new actions to be scheduled may interfere with ongoing and already scheduled operations. At a minimum, the new activities are going to require resources. For example, part of the response to a space weather event may be to divert activities currently scheduled on satellites with space weather issues to other assets, but these ground and space-based resources will have other, competing demands and it is important to optimally allocate these assets across all the demands for them.

These demands (often called requests) fall into a number of seemingly disparate categories that nonetheless can be handled similarly. One category involves activities scheduled on the satellite experiencing space weather issues. Actions such as closing shutters, changing operating modes and maneuvering will affect the ability of the satellite to perform its mission, i.e. its current set of tasks. Those tasks may need to be re-allocated to another satellite. The constraints on which other satellites can assume these tasks will be based on timing and line-of-sight (LOS) issues (orbital mechanics). Another category is tasking earth-based sensors (optical or radar) to focus on specific space objects or to search a particular volume of space. Which resources can perform these tasks and when will also be based on timing and LOS issues involving orbital mechanics. Another category is the tasking of space-based sensors to focus on specific objects or areas on the earth, such as to fulfill requests for commercial satellite imagery. Which resources can perform these tasks and when will, again, be based on orbital mechanics. Therefore

[1] Job Title: President; Department Name: N/A; Street Address/Mail Stop: 951 Mariner's Island Blvd., STE. 360, San Mateo, CA. 94404; AIAA Member Grade: Regular New

all 3 categories will tend to involve constraints between a location on the ground and an orbiting object. The different categories of orbiting objects are those in Low Earth Orbit (LEO), Geosynchronous orbit (GEO), and High Earth Orbit (HEO). Within each category the timing constraints are similar. LEO objects orbit several times a day, have relatively narrow (10-15 minutes) time windows with LOS to specific locations on the ground, and are relatively near to those locations. The set of locations that a LEO object has LOS with is small and rapidly changing. GEO objects orbit once per day, meaning they are approximately fixed in longitude. This means that they have LOS to a large, relatively unchanging fraction of the earth all the time, but at a long range. HEO objects tend to have long time windows of LOS to specific locations and the set of those locations changes relatively slowly over time. Sensor or communication demands (requests) typically inherit the constraints for the types of orbiting objects they relate to. Resource management involves optimally tasking specific resources to meet the requests while obeying all of the constraints.

Scheduling problems involving meaningful resource selection are NP-Complete, meaning that the computational time of any algorithm guaranteed to produce an optimal solution is exponential with the number of those decisions. In the case of space mission scheduling, this NP-Completeness has led to several failures to develop software to automate scheduling for a number of reasons. First, it is straightforward to build bad scheduling systems but difficult to build good ones. I.E. there are relatively simple algorithms, such as priority based schemes, that will generate correct schedules but not very good ones in the sense that many requests will be rejected that could have been scheduled with a more optimum allocation of resources. It is relatively difficult to build good scheduling systems because the NP-Completeness necessitates the use of good heuristics

Many previous attempts at AFSCN scheduling use a naïve approach to assigning communication resources to requests and resolving conflicts where the requests are fulfilled in priority order. Some flexibility may exist in the request which will be used if the primary request cannot be fulfilled, but no earlier processed (and therefore higher priority) requests will be reconsidered in order to try to accommodate the current request. In particular, if the current primary request cannot be scheduled, then the flexibility options will attempt to be utilized; but if these too are not successful, the request will be left unscheduled, even if it had been physically possible to shuffle some of the higher-priority constraints to accommodate the current one. Particularly, this can occur when a geosynchronous (GEO) or high earth orbit (HEO) satellite has a high priority. Their relatively high orbits give them more flexibility as to the timing of their communications. If a LEO satellite has a lower priority (and is therefore considered for scheduling later) it is quite possible that the GEO (or HEO) satellite scheduled earlier happened to use an antenna at a time when the LEO absolutely needed it. When the LEO is finally considered for scheduling, its own opportunity will have been used up by the higher priority GEO and therefore it will fail to be scheduled at all. This is true even if the GEO could have been moved to a slightly different time to allow the LEO to be scheduled.

Although a priority-order allocation guarantees that higher priority requests will be fulfilled before lower-priority ones, it is not optimal (and can perform very badly) and under-utilizes important resources. Consider the following extremely simple example, which is therefore easier to use to illustrate this point, where three requests for the same time period have been received, Requests 1, 2, and 3, respectively, where the priority is highest for Request 1 and lowest for Request 3. Two different stations exist, A and B, that happen to be situated to service two of those requests each, based on the orbits. Station A can service one request at a time and Station B can service two. Request 1 references Station A in its primary request and Station B is listed as an alternate. Request 2's service request lists Station B with no alternate and Request 3 lists Station A with no alternate. Request 1, with the highest priority is selected first by the priority-based scheduler and since its primary requested station, Station A, is available, this is what is allocated to it. The priority-based scheduler then considers Request 2, with the second highest priority and selects Station B, since it is the only one that can service Request 2. When the scheduler gets to Request 3, it finds that only Station A could service its request, but it has no more capacity. Meanwhile, Station B has excess capacity. Obviously this is a suboptimal solution, since it was possible, following an algorithm that considered resource contention, to allocate a station to every request.

But perhaps a scheduler could be devised that systematically tried every possible solution and selected the best, and therefore optimal, one. In the example above, the number of possible solutions is 2 choices for request 1 times 1 choice for request 2 times 1 choice for request 3 = only 2 possible solutions. However, consider a set of requests consisting of only 30 simple requests where there are on average 4 meaningfully distinct choices (e.g. different stations, time windows, or frequencies) for each request. This means that there are 30 distinct decisions with 4 choices each so the number of solutions is 4 x 4 x 4.... x 4 = 4^{30} = over a million trillion possible solutions, which is clearly impractical to systematically search. This is the essence of NP-Complete problems. They require good heuristics to solve them in a close to optimal manner and simple methods for ordering the decisions (e.g. ordering the requests) can produce very poor results.

II. General Scheduling Service Concept

We are creating a highly intelligent and flexible scheduling service available through a Service Oriented Architecture (SOA) for a variety of diverse space-related applications which require optimized resource allocation. These applications would provide a description of the scheduling problem that needs to be solved to the scheduling service. This problem definition primarily takes the form of an XML description of the tasks to be scheduled, the resources available, and the constraints. With enough depth and flexibility specified for these descriptions, any scheduling problem can be represented in this way. The scheduling service delivers back to the requesting application an optimized schedule which represents a high-quality resource assignment solution to the problem. The scheduling service stores within its local database every problem and solution that it has solved. This way incremental scheduling requests from clients can come to the scheduling service, merely referencing the previous problem and solution and only needing to specify changes.

The scheduling service receives these scheduling requests from a variety of SOA clients. For example, one application, receiving commercial requests for satellite imagery, would take these requests and the satellites under its control and transform these into tasks (or requests), resources (the satellites and their payloads), and visibility and other constraints, and pass the resulting scheduling problem to the scheduling service which would return an optimized set of observation assignments. Another application, relating to reassignment of tasks from satellites under space weather induced duress, would combine the current tasking of the satellites experiencing difficulties with the tasking of the satellites which could potentially assume some of that tasking, and send the resulting scheduling request to the scheduling service, which would respond with the optimized reallocation of all the tasks to the remaining satellites. A third application would the set of observation requests for a 24 hour period for the tens of thousands of objects in the space catalog, primarily consisting of manmade debris orbiting the earth. The scheduling service would return the set of assignments of observation tasks to radar and optical sensor resources.

III. SOA Integration

The general scheduling service was prototyped using the Java Messaging Service. The Java Messaging Service is a messaging standard that allows for broadcasting (using topics) and point to point communication (using queues) and is used by many SOAs. The current Scheduling Service prototype can be accessed as a service via the Java Messaging Service. The process for using the current Scheduling Service prototype as a service is as follows:

1. The client creates an xml representation of the scheduling problem according to the data model described below.
2. The client sends the scheduling problem and a return address (queue) to a permanent Java Messaging Service topic that is monitored by the scheduling service.
3. The service receives the scheduling problem, schedules, and returns the solution using the same data model to the client via the return address provided by the client.

The current demonstration of the scheduling service prototype uses a client based on an existing scheduling tool GUI for schedule problem definition, viewing and schedule editing. I.e. the user defines the problem and views the solution in this GUI, but the actual scheduling is accomplished by sending the scheduling problem in XML format via the Java Messaging Service to the scheduling service and receiving the solution in the same way.

For the current prototype, scheduling problems and solutions are specified with an xml schema described here:

1. Task: A task is some action to be scheduled, like a sensor observation, communication pass, or tracking episode. Tasks have properties like duration and the earliest and latest possible time to do the task. Tasks often require the use of one or more resources.
2. Resource: A resource is some personnel, equipment, service, etc. that is required for the completion of a task. Resources might be usable only at certain times (LOS for a satellite, work hours for personnel, etc.)

as described by a calendar. Resources may be individual (usable by only one task at a time) or continuous (usable by multiple tasks simultaneously.) Resources also have a quantity indicating how many or much of them there are.

3. Calendar: A calendar specifies the set of times that a resource or task can be scheduled.
4. Constraint: A relationship between tasks and/or resources that specifies some constraint on the selection of the time or resource for the task. This might be a temporal requirement in the execution of the tasks such as one task having to complete before another can start, that two tasks should use the same specific resource, or that a task can only use a resource that has the proper visibility at the time the task executes.
5. Resource Set: A collection of resources that can be used interchangeably for some purpose.
6. Resource Requirement: Specifies what combination of resources could be used to complete a task. E.g. satellite communication pass might require an antenna and the correct type of service.
7. Resource Assignment: The set of resources actually scheduled to be used from among the options provided in a Resource Requirement.
8. Visibility File: The XML schema also references the name of a file that stored previously calculated visibility opportunities between objects in space and locations on the ground. If applicable, it can also store visibility between two different objects which are both in space.

The scheduling service receives these scheduling requests from a variety of space mission clients. For example, one application could involve tasking space-based or ground based sensor resources. It would transform the needs for sensor information into tasks (or requests) for sensor resources, and pass the resulting scheduling problem to the scheduling service which would return an optimized set of sensor assignments. Another application, relating to reassignment of tasks from satellites under attack or duress would combine the current tasking of the satellites experiencing difficulties with the tasking of the satellites which could potentially assume some of that tasking, and send the resulting scheduling request to the scheduling service which would respond with the optimized reallocation of all the tasks to the remaining satellites.

IV. Multiple Schedulers

Since, like most NP complete problems, generating a correct schedule is much harder and more time-consuming than testing or grading one, it would be possible for the scheduling service to include several different scheduling algorithms, all of which were independently generating schedules. Relatively simple and quickly executing grading software could then evaluate each schedule and select the best one. One algorithm might be the bottleneck avoidance algorithm (including swapping out earlier processed, lower priority tasks), described later. The other might be a more conventional priority based scheme which was guaranteed to service higher priority tasks over lower priority ones, which is useful when the number of required tasks far outweighs the resources available to execute them. In situations such as the simple example shown above where it is possible to service all tasks with proper consideration of resource contention, the best solution might be from the bottleneck avoidance algorithms. But there may also be some situations when all service requests cannot be met where a priority-based algorithm performs better in terms of delivering an acceptable solution quickly.

A preprocessor component could, among other things, examine the details of the incoming scheduling request, decide on the appropriate scheduling method or methods to call, grade the responses if more than one is returned, and select the best one. In this way emergency service requests requiring attention immediately would be passed to a very high speed (but less optimal) Emergency Response Scheduler for immediate scheduling, so that the appropriate tasks can be passed back and executed immediately. Presumably this would only contain a relatively small number of requests that require immediate execution. Otherwise, or for the remaining requests, the Preprocessor selects one or more appropriate Subschedulers. These Subschedulers will all be independent, and therefore would easily fit into a distributed architecture with each Subscheduler having its own computing resources, if required.

V. High Level Architecture

The High Level Scheduling Service Architecture is shown in the following figure. Applications integrated in the Java Messaging Service SOA with scheduling problems to be solved send an XML description of the scheduling problem (Tasks, Resources, & Constraints) along with visibility data and scheduling method selections through the Java Messaging Service interface to the scheduling service. The preprocessor calls the appropriate subscheduler, implemented in a generic, intelligent scheduling system architecture.

These Subschedulers are the primary focus of the research and development and are the components that must address the really difficult aspects of space mission resource assignment and scheduling problems. Each Subscheduler conceptually takes as input the tasks, resources and constraints and produces, as output, the assignments to the tasks of resources and time windows that meet the constraints. Each task description includes what resources it requires and for how long, and the constraints that it is involved in. If tasks are hierarchical, each task will list its component subtasks. In practice, most of the resources needed by tasks will have been predefined, with only updates as to their availability being passed in (e.g. sensor failure) at scheduling time. The Scheduler generally maintains its own record of the availability of each resource based on the tasks and the time windows currently assigned or tentatively assigned to them. The constraints may be hard (an absolute requirement) or soft (satisfaction is desired but not absolutely necessary) and may be temporal, resource-oriented, or of miscellaneous type. The applications requesting scheduling services can also optionally pass in constraints on the scheduling process itself, such as how much time is available, which specific subscheduler should be used or which sets of methods should be used for the decision points, some of which are described below.

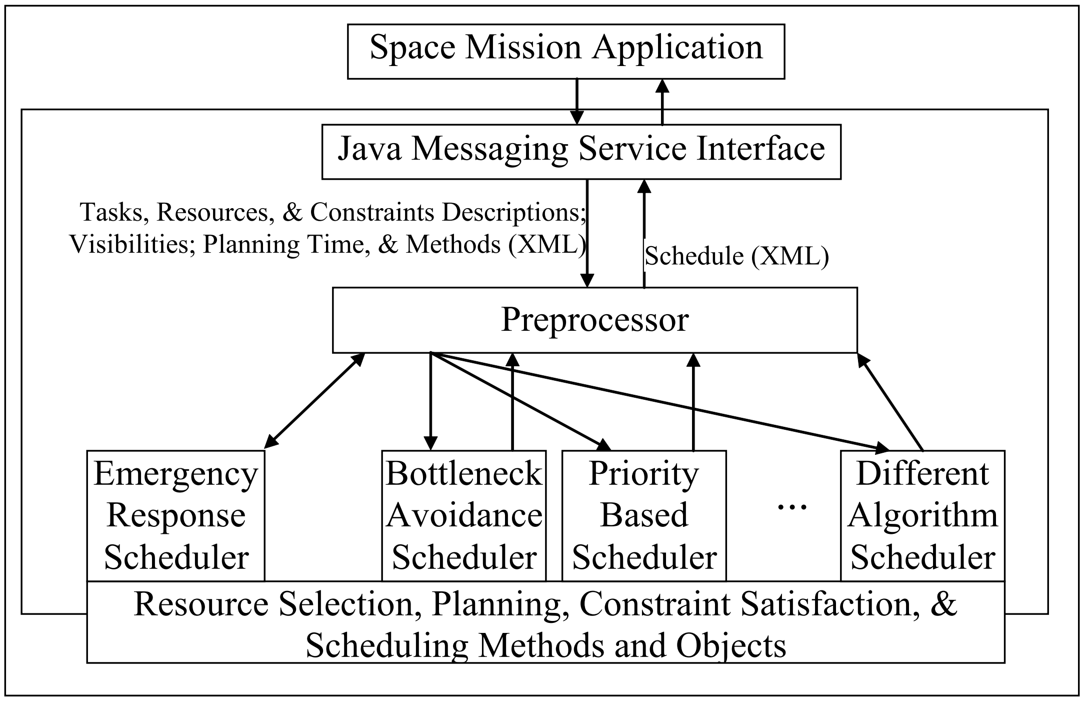
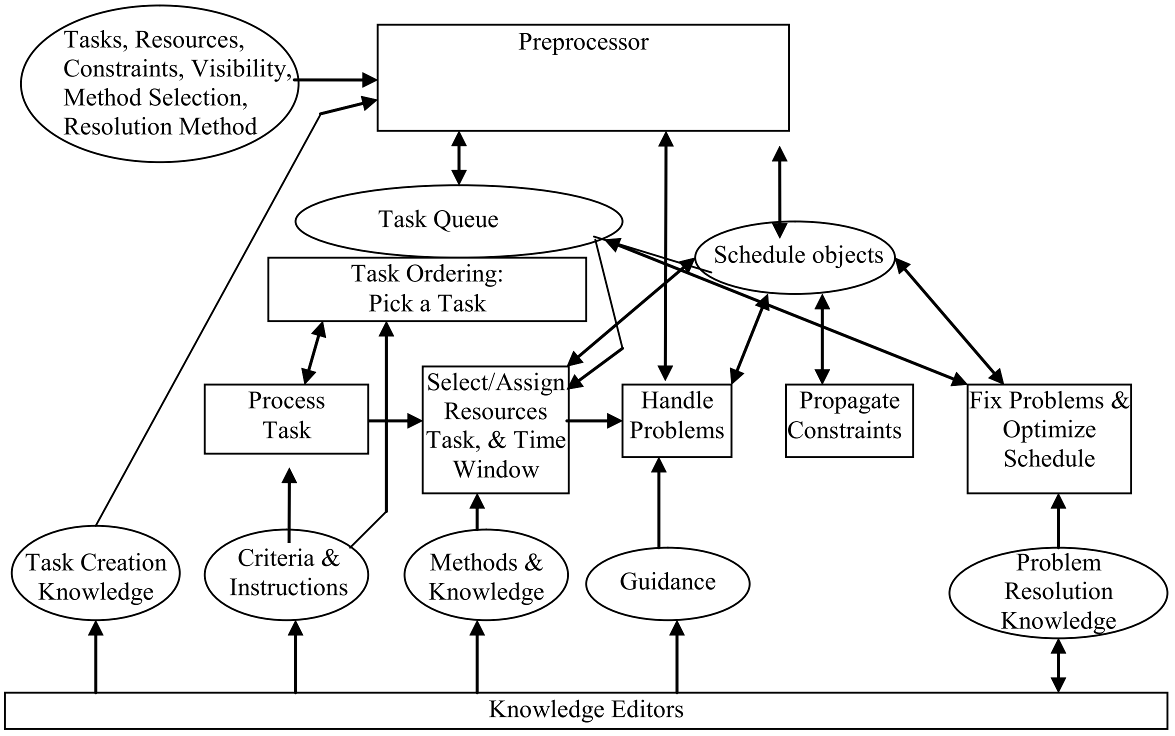

Space Mission Application
Java Messaging Service Interface
Tasks, Resources, & Constraints Descriptions; Visibilities; Planning Time, & Methods (XML)
Schedule (XML)
Preprocessor
Emergency Response Scheduler
Bottleneck Avoidance Scheduler
Priority Based Scheduler
...
Different Algorithm Scheduler
Resource Selection, Planning, Constraint Satisfaction, & Scheduling Methods and Objects

Figure 1. High Level Scheduling Service Architecture.

Each Subscheduler has several decision points, each of which our generic intelligent scheduling architecture allows to be customized through selection of existing options or plugging in new software. One major set of decisions is what the scheduler should process and in what order to process. One common solution is to process tasks in due date or priority order. Another possibility is to process resources in the order in which they become available. Still another is to process constraints, the tightest ones first. We expect, as described in the algorithm description below, to process requests, sorted so that the tasks that request the most constrained resources at the most constrained times are scheduled first. It is extremely important to note that the order in which processing will occur has a very large effect on the quality of the solution and that simple priority or time ordering schemes will often produce very poor results in any reasonably complex scheduling situation.

Another major decision point is assigning to a task the needed resources and time windows, while satisfying the applicable constraints. Usually the time window is constrained by the need to get a task done by a certain time while having to wait for events or other tasks to finish before they can start. There may be an additional soft constraint to have the task finish as soon as possible or to start as late as possible. Usually there is at least some freedom in choosing the time window at the same time that the specific needed resources are selected. Choosing the set of resources that are available at the same time that best meet the constraints, and that are best for the overall plan can be difficult. "Greedy" systems that choose what is best for the specific task or resource often produce poor results, overall. Some systems use a resource balancing approach whereby the least utilized resource is chosen, under the assumption that if a later-processed task needs a specific resource instance, all of them will still be available; but this can also run into problems, depending on the processing order, especially in cases like space mission applications where resources are expensive and therefore likely to be tight. A better approach is to choose the resource that other tasks are least likely to need. This is the approach in the algorithm described below.

One general domain-independent resource constraint satisfaction algorithm that we have developed was inspired by Sadeh[1]. In this algorithm, first, each task's resource time requirements are spread evenly across all of each possible resource's possible time windows, without regard to when other tasks might be eventually scheduled. These individual profiles are summed across all tasks for each resource. This first step is done to determine which resources are most overburdened and at which times. This is used to determine which tasks should be scheduled first and which time/resource windows should be avoided.



Tasks, Resources, Constraints, Visibility, Method Selection, Resolution Method
Preprocessor
Task Queue
Schedule objects
Task Ordering: Pick a Task
Process Task
Select/Assign Resources Task, & Time Window
Handle Problems
Propagate Constraints
Fix Problems & Optimize Schedule
Task Creation Knowledge
Criteria & Instructions
Methods & Knowledge
Guidance
Problem Resolution Knowledge
Knowledge Editors

Figure 2. Internal Scheduler Architecture.

Often, this algorithm generates an optimal plan in one pass. It accomplishes this by picking the most overburdened resource, that resource's most overburdened time, that time's task which most contributes to the problem but which has the least flexibility, and finally by choosing a resource and time window that minimizes the degree of overburdening most. The intuition is to concentrate on the most contended for resources at the most contentious times and then to concentrate on the tasks with the least flexibility, since this lack of choice could cause problems later. This might occur if a task with more flexibility was scheduled for a resource at a time where another task absolutely needed it. Finally, the algorithm attempts to make time window choices that minimize the degree of overburdening.

One major decision point arises when, during the assignment of resources and time window to a task, no reasonable assignment can be found. This may involve unsatisfiability of a soft constraint, in which case it will most likely be ignored, but may also involve the inability to satisfy hard constraints or to find needed resources at the required time. In one of these latter cases, the appropriate action for the planner depends on the aspects of the domain, the way it is modeled, the scheduling time available, and the optimality of the other algorithms in use. For example, if an unavailable resource is not completely saturated, it may be appropriate to try to shuffle some of the other task assignments to make a "hole" in a resource availability profile for use by this task. Perhaps a lower priority or less important task that is already scheduled should be unscheduled to make room for this task. The concept of changing the plan in the same cycle brings up the notion of backtracking. A small amount of backtracking can be important to optimize a plan in relatively small ways, but over-use of this technique degenerates into systematic search, which should be avoided for most complex problems. Often heuristics can be efficiently used to handle the problem of resource/time window unavailability. These take two forms. One is that often some of the supposed hard constraints or resource requirements have some leeway. For example a sensor setup task may be specified to take 60 seconds but in an extreme situation, 30 seconds might be acceptable if the only alternative is to not have the sensor event occur at all. Other examples are tasks that may occur somewhat in parallel rather than in series as specified (again, versus not happening at all). For example, an intelligence gathering request might specify that they would prefer time between the passive radar search for a jammer and the task to collect overhead imagery in order to more precisely determine its location before taking the image. However, if the image is large enough to definitely include the jammer (albeit at lower resolution) it might be acceptable to perform both sensor tasks concurrently. This would be an especially relevant example if the particular satellite with these sensors was in serious contention at this time. The other form of heuristic is one whereby all resource requests and constraints can be met by a clever trick that the human planner has found to be useful in similar situations; e.g., human planners have found that GEO and HEO related requests are the most likely to be able to be moved to make room for another, more tightly constrained LEO related request. (Of course our proposed resource assignment algorithms already take this into account the first time through.) These latter forms of heuristics are less likely to be useful if the ordering and selection algorithms are near optimal. Ultimately it may be necessary to place the task in question (or another one swapped out to make room for it) on an unschedulable list, to be dealt with at the end of the cycle.

After each item has been processed, another decision point is reached, that of optimizing the plan just produced before the resource assignment cycle ends. This includes trying to solve the remaining problems and optimizing the soft constraints. Recall that it is possible that acceptable resources and time windows could not be found for some tasks. Thus, tasks may be on an Unscheduled List or may be scheduled in such a way as to create conflicts and otherwise violate constraints. A Case-Based Reasoning (CBR) module exists to suggest conflict resolution strategies based on previous similar conflicts. These may be able to be applied by the scheduler itself to fix the conflict or may require user approval. The use of CBR for Conflict Resolution is discussed more thoroughly in Stottler[2]. Alternatively policy rules may dictate how to resolve conflicts, such as swapping out lower priority requests to allow higher priority ones to be satisfied.

The scheduling service will operate on a planning cycle concept to handle evolving situations. A preprocessor will examine the input scheduling problem and planning time available and determine if the emergency or other very rapid subscheduler should be called to schedule any tasks immediately for immediate execution. This can be followed by additional requests scheduled with more planning time and deliberation to achieve a more optimum use of the remaining resources. So urgent tasks can be scheduled and executed during the more normal planning cycle of the majority of the tasks. This hybrid approach achieves the benefits of both approaches – high speed scheduling and execution when necessary and more optimum scheduling and use of resources when more time is available to develop a more optimum schedule.

Under more normal circumstances, the preprocessor takes the updated scheduling problem and the available planning time and chooses one or more from among a small number of planning options, each based on a different algorithm, having different trade-offs as to optimality and run time. These options will include a systematic search algorithm which was guaranteed to produce an optimal result but could only be executed with a very small number of tasks. One option would be an algorithm applicable to a large number of tasks, but within the design parameters of the resources in question. This algorithm would be heuristic in nature, to avoid the exponential computational time associated with NP-Complete problems, but would produce superior solutions to the task scheduling and resource assignment. An additional algorithm, if determined to be necessary, will be one to handle the case of an overwhelming problem, where far fewer resources are available than required for the requested tasking. This approach also makes it straightforward to add additional planning algorithms, perhaps because they have been shown to be more optimal in specific types of situations. With a simple grading scheme (such as the summation of priorities of fulfilled tasks) the best resulting plan can be chosen from multiple independent algorithms. This might be especially appealing with the availability of additional processors.

VI. Results and Future Work

In addition to prototyping the scheduling engine as a JMS accessible service in an SOA, the scheduling engine has itself been applied to several different space mission scheduling applications. The general scheduling service has been used to first prototype an automatic scheduling system for a DOD satellite communication network and is now being used as the basis for the development of an operational system. The scheduling engine discussed here has also been applied to develop an automatic scheduling system for a NASA space communication network and for a NASA application scheduling in-space resources during manned missions. Finally, this service is being applied to the scheduling of the sensor resources of the Space Surveillance Network (SSN) to schedule the tracking and signature gathering tasks to maintain and expand the Space Object Catalog. In all cases, the system described here has performed as well as or better than the humans performing the activity.

VII. Conclusion

Our work has shown that it is possible to define a general scheduling service which can be successfully applied to a variety of diverse domains. This includes the development of an XML schema which can represent both the detailed, rich description of scheduling problems to be solved and the resource assignments and accompanying time windows that constitute the solutions. Furthermore, by including a variety of different scheduling methods, built in an architecture which allows each scheduling system decision to be easily customized, a wide diversity of space scheduling applications can accommodated.

References

[1] Sadeh, N., “Lookahead techniques for micro-opportunistic job-shop scheduling,” Ph.D. Dissertation, Carnegie-Mellon Univ., Pittsburg, PA, 1991.

[2] Stottler, R., “Satellite Communication Scheduling, Optimization, and Deconflcition Using Artificial Intelligence Techniques,” *AIAA Infotech@Aerospace 2010*, Atlanta, GA, 2010.