Can Customers Schedule Their Own Payload Activities?

John Jaap
John.Jaap@msfc.nasa.gov
256-544-2226

Elizabeth Davis
Elizabeth.Davis@msfc.nasa.gov
256-544-2227

Mission Support Systems Group
Ground Systems Department
Flight Projects Directorate
Marshall Space Flight Center, AL 35812
National Aeronautics and Space Administration

Abstract

The term "customer" in the title refers to the payload developers; they are the *real users* of a space vehicle, after all. The answer to the question lies in the ability to design and deploy a system that allows multiple simultaneous users to schedule activities that require shared resources. In addition, the system must be designed so that it can easily be used by a community whose members, while being experts in their payloads, know little or nothing about scheduling. An effort is underway at Marshall Space Flight Center to demonstrate the feasibility of allowing users to schedule their own payloads. A web-based request-oriented scheduling engine and the infrastructure to support it are being investigated. This system will allow multiple users, each at a personal computer with a web browser, to formulate scheduling requests and submit the requests for immediate automatic scheduling.

Introduction

Scheduling payload operations has historically been done by a cadre of mission planners who act as agents for the payload developers. The members of this cadre are experts in the features, capabilities, limitations, and language of the scheduling engine. Different members of the cadre have different scientific and technical backgrounds; they are usually matched to the payload they represent. The members become experts in the payload that they represent – often with the help of extended input from the payload developers.

The process of collecting requirements is often protracted with multiple iterations. The payload developer submits payload requirements and descriptions in the requested format. A cadre member reviews the information and contacts the payload developer for a better understanding. The payload developer modifies his submission.

After the cadre has an adequate (in-depth) understanding of the requirements, the cadre members use the scheduler to produce a timeline. The payload developers review the timeline, make comments, and the cadre makes repairs to the timeline. Usually, several months elapse between the requirements submittal and the completion and publication of the timeline.

The scheduler currently used for the International Space Station payloads requires expertise to represent the payload requirements. The lack of several key features causes the cadre to spend a great deal of time describing a payload's actual scheduling requirements in the "vernacular" of the scheduler, and then to hand build the timeline with a manual timeline editor to get the desired results. The scheduler cannot represent or process optional sequences of operations (multiple scenarios), choice of constraints within a non-homogeneous group, soft (fuzzy) requirements, or resource lock-in across sequential tasks. Furthermore, the current scheduler only supports one cadre member at a time, so the cadre members must either take turns or funnel all scheduling through one member.

If a scheduling system existed that allowed payload developers to schedule their own payloads, a better timeline could be produced in less time with fewer cadre members supporting the system. The delay between describing and submitting a scheduling request and viewing the results would be reduced to minutes rather than months. The payload developers could then refine and resubmit their requirements until they got a satisfactory timeline. The cadre would only have to preload the scheduling engine with the envelopes and allocation constraints, define the station and payload equipment resource requirements, and post-check that the timeline is safe and meets programmatic constraints.

ROSE The Mission Support Systems Group at MSFC has embarked on an effort to design and demonstrate (and possibly deploy) a system that ameliorates many of the shortcomings of the current system by allowing the payload developers to schedule their own payload activities. It includes a graphics-based method for formulating scheduling requests and a centrally located, multi-user Request-Oriented Scheduling Engine (ROSE) working against one current set of resource availabilities and one current timeline. The system uses the World Wide Web to allow the user community (the payload developers – the customers) to readily access the system from a personal computer or workstation. The name ROSE is being applied to both the project as a whole and to the scheduling engine itself.

Web-Based Architecture

ROSE is a web-based application. The users access the system via the World Wide Web; no ROSE-specific software is installed on a user's computer and no data is stored on a user's computer. The user navigates via a web browser to the ROSE web site, logs on, and proceeds to formulate scheduling requests and submit them for scheduling. The user can also review the in-work timeline, delete tasks from the timeline, and view ancillary information such as resource allocations and envelopes. An overview of the ROSE architecture is shown in Figure 1.

A web-based system is ideal for the payload developers. Maintenance is literally zero. Opening the web page with a browser will download and execute the latest version of the client without any effort on the user's part. The user can switch between client computers in his office, home, or even use a portable without installing ROSE-specific software or moving any files. Security is the only concern, and security is being integrated into the ROSE research and development effort so that an adequate solution can be deployed.

Implementation The user of ROSE initiates a "web session" by opening a web page on a remote web server. The web server is Microsoft's Internet Information Server (IIS) running on a Windows NT/2000 server. The web page downloads and runs a Java applet. Logon is handled by the Java applet; the web session is managed by IIS via Active Server Pages (ASP) and a global.asa file. All communication with the Java applet, including access to the database, is facilitated by a set of ASPs written in Visual Basic Script. Sessions are terminated when the user goes to another web page or exits the browser, when the server does not receive communication from the client for an extended period of time, or when the client or the server is restarted. The client automatically initiates a new session, without user intervention, when needed. The session manager, which handles locking of the data in the database, prevents an account from having more than one active session. This implementation does not require a continuous connection between the client and the server. It even allows the client computer to be disconnected from the network for an extended period without loss of cached data or edits.

Modeling – The Critical Element

Modeling in the context of activity scheduling has historically meant defining an activity's requirements for shared resources (power, crew, etc.), time-dependent constraints (when the vehicle is within view of a target), and sequencing of activities relative to each other (warm-up before data-take before shutdown). In the context of ROSE, modeling is also called "request formulation," because a model is the core of a scheduling request.

Modeling is the process of representing the requirements in a manner usable by the scheduling engine so that it can produce a correct and acceptable timeline. Modeling always requires an in-depth understanding of the payload and how the scheduling engine behaves. In the current operations concept, a payload developer provides an in-depth description of the payload to the group, called the scheduling cadre, which builds the models and eventually runs the scheduler. After several extended dialogs with the payload developer, the cadre builds the models for the payload, produces a timeline, and presents it to the payload developer for review. The payload developer reviews the timeline and requests changes; the cadre tweaks the models and (after sufficient iteration with the payload developer) produces an acceptable timeline.

If payload developers, who already have expertise in the payloads, are to formulate scheduling requests and submit them to a scheduling engine, they must have expertise in the scheduling engine – *this is critical*. Being an expert in the behavior of the scheduling engine means knowing how the engine will react to a given model, and how to build a model to achieve the desired results. ROSE will make the payload developers virtual experts in using the scheduling engine by making them experts in modeling. ROSE uses a request format that is a natural representation of the requirements without adding artificial constraints or constructs. We call this modeling methodology *high-fidelity* or *hi-fi* modeling – the model looks like the real world payload and the engine interprets the model as expected. ROSE provides immediate feedback (when a request is submitted, the resulting timeline is available immediately), thereby exposing users to the workings of the scheduling engine. After only a few submittals, the users will know what to expect from the engine when a request is submitted; i.e., they will become virtual experts.

In addition to supporting "hi-fi" models, ROSE provides a user-friendly interface that is neither laborious nor time consuming and which requires little or no training. The user interface of ROSE is the same as that of the Interim User's Requirements Collection (iURC) system currently being used to collect payload scheduling requirements for at least the

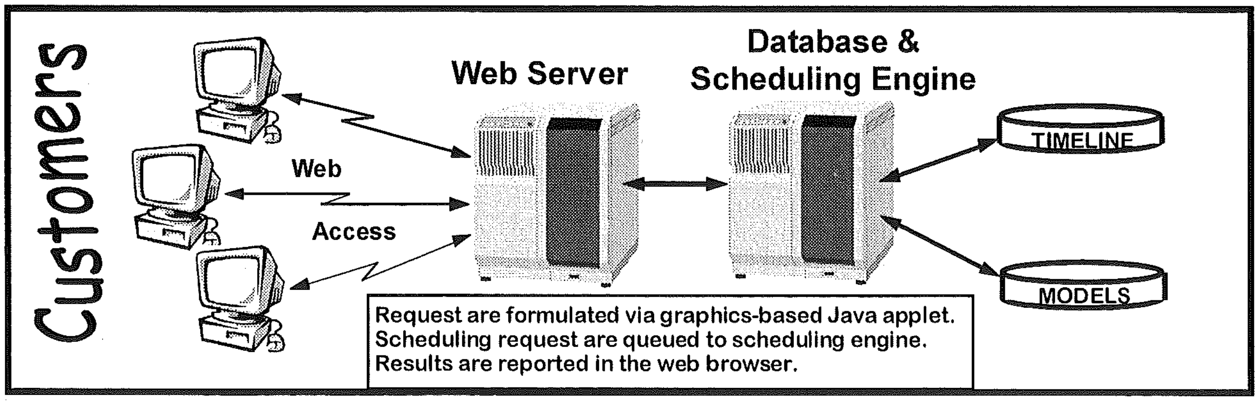
Customers
Web
Access
Web Server
Database &
Scheduling Engine
TIMELINE
MODELS
Request are formulated via graphics-based Java applet.
Scheduling request are queued to scheduling engine.
Results are reported in the web browser.

Figure 1. ROSE Architecture Overview

first four increments of the International Space Station. This interface is described in a previous paper (Jaap, Davis & Meyer, 1997) and in the on-line user's manual for iURC. The modeling process *itself* employs graphical methods to describe activities and sequences. The user is presented with

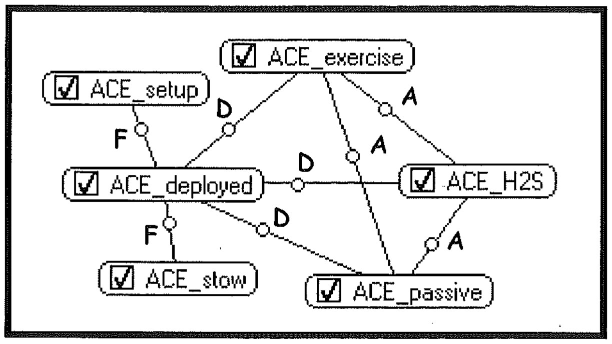

Figure 2. Typical Sequence

a canvas to which items are added and arranged in a hierarchy (for activities) or in a network (for sequences). Details of the requirements are entered via dialog boxes. The advantages of using a graphics method are best illustrated by example. In the typical International Space Station sequence shown in Figure 2, the temporal relationships marked with an F are "follows" relationships, those marked with a D are "during," and those marked with an A are "avoid." The sequence indicates that ACE_setup is followed by ACE_deployed, which is followed by ACE_stow; during ACE_deployed, ACE_H2S, ACE_passive, and ACE_exercise are done, but while avoiding one another. ACE_H2S, ACE_passive, and ACE_exercise are themselves sequences. This example is a simple sequence; International Space Station users frequently submit sequences with twenty or more tasks and many relationships, including some to station tasks like reboost and shuttle_docking. The graphics representation is understandable even when extended to large and complex sequences. The modeling methodology supports optional arrangements of tasks within the sequence (sequence scenarios), soft requirements, and choice of constraints within a group with lock-in across sequence members.

Implementation The Java applet provides both request formulation (modeling) and request submittal. Models are stored in Microsoft's SQL Server database hosted on a computer at the site of the web server. When the user selects a model or a portion thereof for editing, it is downloaded from the database via the web server (shown on left side of Figure 3). Edits to the models are posted to the database via the web server whenever the user selects a different activity or sequence, submits a scheduling request, or makes an explicit request to save the data. When posting updates, database stored procedures are used (shown on right side of Figure 3).

A model is pre-checked for errors before it is submitted to the scheduling engine; or a user can request a pre-check at any time. When a sequence is pre-checked, all the referenced sub-sequences and activities are also checked. Checking is done on the web server computer by an ISAPI (Information Server Application Programming Interface)

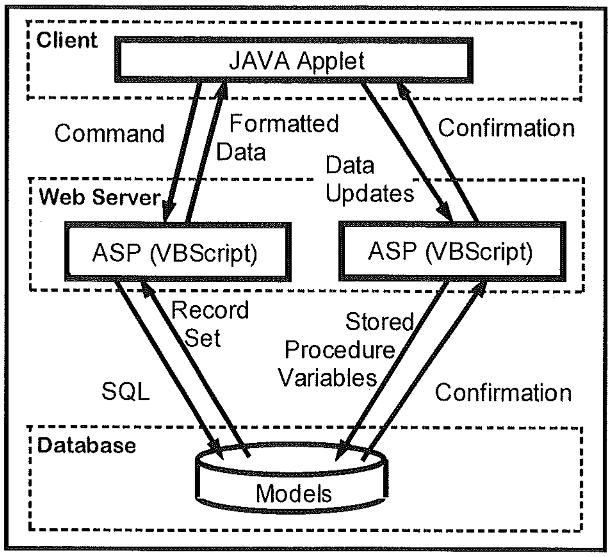

Figure 3. Reading and Writing

extension to IIS (shown in Figure 4). The ISAPI is written in C++. Since the ISAPI is multithreaded, one instance services multiple simultaneous users. A separate database connection is made for each pre-check request.

There is no explicit configuration control of the user's models. A user can always edit his models. However, when a model is added to the timeline (scheduled), an instance of

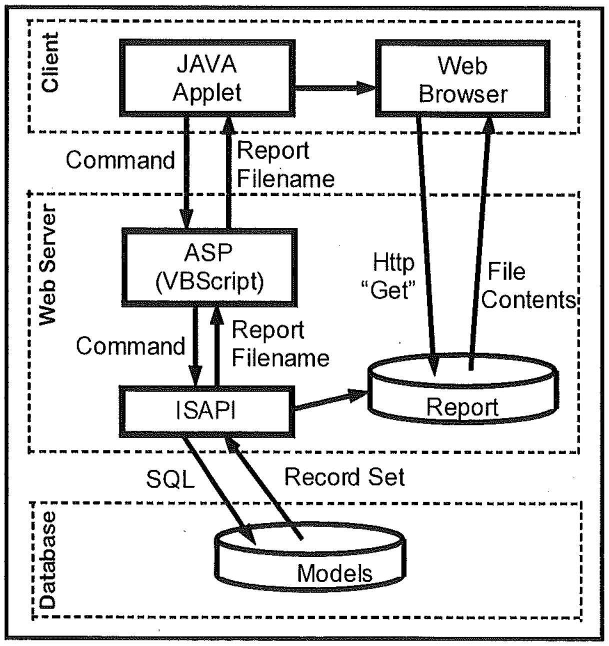

Figure 4. Checking a Model

that model is saved with the timeline and never edited. Thus, editing a model does not edit what is already scheduled, and configuration control of users' models is not required.

Scheduling

While the scheduling engine is the key element of the system, the user should not need to know how it works – the user needs only to hone his modeling skills. The scheduling engine must understand the "language" of modeling perfectly and should behave exactly as expected. Moreover, the scheduling engine must provide feedback (reports for successes and explanation for failures) to help the user improve his modeling skills. To be useful, the ROSE system as a whole must respond to scheduling requests from each of its multiple users in only a few minutes.

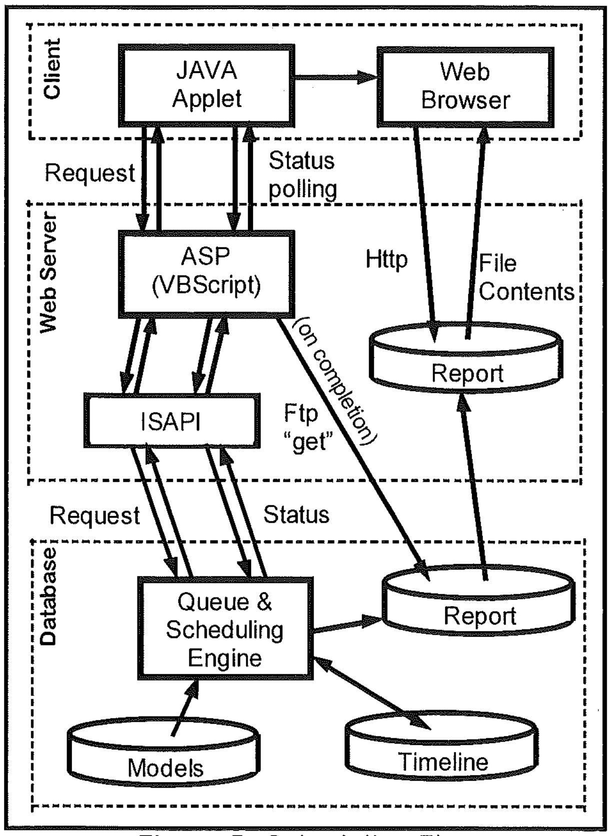

Figure 5. Scheduling Flow

In ROSE, each scheduling request initiates an attempt to schedule one performance of one sequence that may have embedded sub-sequences, repeated tasks, and multiple scenarios. The scheduling request is primarily the data in the model, but the user can override or add additional constraints such as a scheduling window, starting time frame, and scenario specification.

Due to the nature of ROSE, some characteristics of the scheduling engine are compulsory. It must handle the models. It must respond quickly. Everything it schedules must be "valid;" i.e., resources are never oversubscribed and constraints are never violated. Once something is in the timeline, only the user can remove it; the scheduling engine doesn't adjust one task to fit another task into the timeline. It must constrain each account so that its resource allocations are not exceeded. Furthermore, ROSE must not lose anyone's scheduled tasks if the scheduling engine or the computer crashes – scheduled tasks are committed to the database immediately after scheduling

Implementation The Java applet provides the user interface for submitting a sequence to the scheduling engine. When the schedule button is clicked, the model is saved to the database via the web server and is then pre-checked for errors. If the model has no errors, a dialog box permitting the user to make selected model overrides is presented. When the user completes the dialog, the request is sent to an ASP on the web server. The ASP passes the request to an ISAPI that forwards it to the scheduling queue. Only a single instance of the multi-threaded ISAPI runs on the web server; it funnels scheduling requests from all users into the head of a communications pipeline to the scheduling queue at the front end of the scheduling engine. After a request is queued, the applet regularly polls the scheduling engine, via an ASP and an ISAPI, to get the status of the request.

The scheduling engine is a Windows NT/2000 "service" with a control console for setup. The scheduling engine accesses the model, allocation, and timeline databases to determine when, in the timeline, to schedule the request. When the request is scheduled, a report is written on a local file in html format. If the request cannot be scheduled, an explanation is written on a local file in html format. The request is now satisfied, the entry is cleared from the queue and processing begins for another request. A summary of the results is returned to the applet when it next polls the status.

Upon receiving the notice of completion, the Java applet sends a directive to the user's web browser to display the report. For successful requests, the report web page contains a form that is preloaded with the request to delete what was just scheduled. Figure 5 shows the main information flow for a scheduling request.

When a request is scheduled, a copy of the model is stored with the timeline, thus allowing the user to continue editing the model without limitations.

Inspecting the Timeline

Implementation A web-page form provides the user interface to request a display of timeline segments. The user can specify the start and stop times of the report and that only the user's data should be included. The request is sent to an ASP that converts it to a SQL statement that is sent to the database. The resulting record set is reformatted into an html page by the ASP and returned to the web browser. Figure 6 shows the flow of information when inspecting the timeline.

By the time ROSE is deployed, inspection may be based on XML (extensible markup language), with the ASP returning XML data and an XML control in the web browser handling the display of data. Using XML will provide client-side control of what is displayed, thereby significantly increasing the responsiveness of timeline inspection and reducing the processing load on the web server and the database.

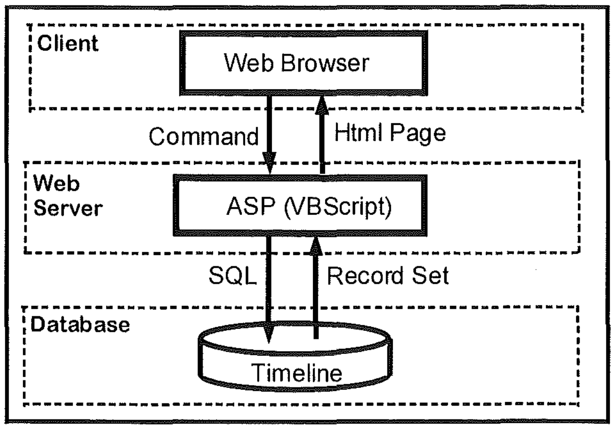

Figure 6. Inspecting the Timeline

Deleting from the Timeline

Implementation The timeline inspection web page provides the user interface for deleting tasks from the timeline. The user selects a sequence performance to delete and sends the request to an ASP on the web server. The ASP forwards the request to the same ISAPI that handles scheduling requests. The deletion request is put into the scheduling queue for processing by the scheduling engine; deletion requests are processed before scheduling requests. Figure 7 shows the flow of information when deleting from the timeline.

Manual Editing of a Timeline

The ROSE architecture does not support true manual editing of a timeline by remote users. A simulation of timeline editing can be accomplished by adding a command to reschedule (delete a specified sequence performance and schedule a performance of a sequence), with its supporting user interface. If the requested sequence could not be scheduled, the original sequence performance would be restored.

Other Applications

Replacement for Current Scheduler ROSE could be used as a replacement for the current scheduler with significant benefit. It would eliminate several of the shortcomings of the current scheduler. However, this approach does not make the payload developers experts in modeling because it does not provide the necessary immediate feedback. The scheduling expertise still exists only in the cadre, and they still have to become near-experts on the payloads they schedule. This shortcoming could be overcome by allowing the payload developers limited access to ROSE.

Standalone What-If Scheduler Payload developers could use ROSE to do "what-if" scheduling. If ROSE were also the scheduler used by the scheduling cadre, then payload developers could become modeling experts; and, by providing usable models, eliminate the requirement for the scheduling cadre to become experts on the payloads.

Job-Jar Scheduler ROSE could fill the need to have a scheduler that allows the crew of the International Space Station to schedule the "job-jar tasks." These are tasks that have been defined by payload developers, but not scheduled; instead, they have been put into a collection of tasks that the crew can do at their discretion. Currently, only tasks that do not utilize shared resources (other than crew) are candidates for the job jar because there is no way for the crew to know payload resources requirements and timeline availabilities. ROSE could easily fulfill the requirements of a job-jar scheduler.

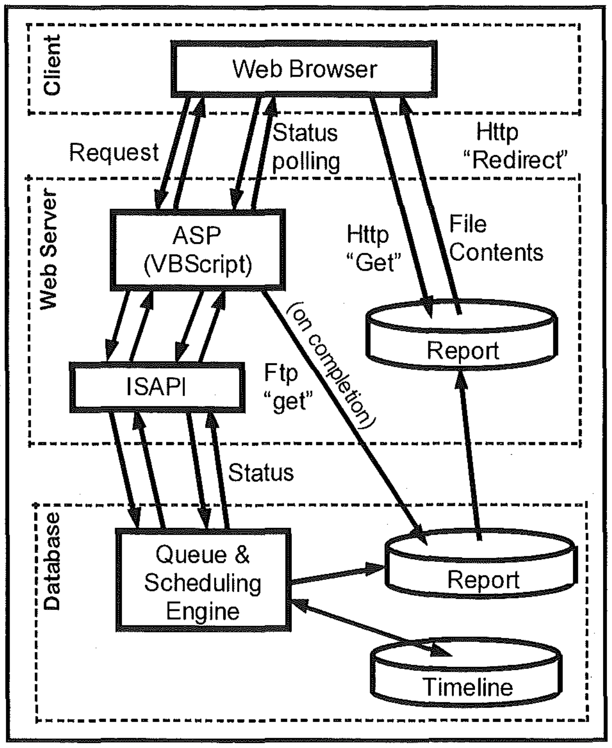

Figure 7. Deleting from Timeline

Summary

Can payload developers schedule their own payload activities? The answer is *YES.* A system is being designed and demonstrated that will provide all the necessary features to support this new approach to payload operations. The key is high-fidelity modeling and a scheduling engine that can schedule the models. The modeling challenge has been met, and the solution is operational in iURC. Feedback from the payload developers using iURC indicates that this modeling approach meets the objective of being able to represent even complex requirements in a straightforward, easy-to-use manner. The World Wide Web provides the needed remote access so that the payload developer community can access ROSE with ease. Standard web software languages and packages provide or enable most of needed features of ROSE.

Status of Research The ROSE project is a research and development effort to investigate and demonstrate a system that addresses all technical issues necessary to allow payload developers to schedule their own payloads. To date, extensive work has been done on the critical element of modeling and it has been put into operation via iURC. The system architecture has been designed and demonstrated. A prototype of the scheduling engine has been developed, but it needs some rework to exactly match the modeling. Security issues are being investigated, but features such as firewalls are standard fare and will not be demonstrated. An end-to-end demonstration with a stubbed-out scheduling engine has been done.

Security Security of the ROSE architecture is a major concern. ROSE is a web-based system that directly affects spacecraft on-board operations. Therefore, security remediation has been integrated into the research from the beginning. Since ROSE may become a reality, it is not wise to reveal the tailored safeguards which are being developed. However, some of the standard safeguards that might be included are firewalls, perimeter security of the host computers, stringent password rules, challenge-response recognition of users, client computer certificates, address recognition of client computers, and secure socket layer communication.

Paradigm Shift Letting payload developers schedule their own payload activities is a paradigm shift for NASA. While it is clear that the ROSE approach will provide better customer satisfaction and that cost savings could be realized, a solution to the programmatic issues has not been embraced. How will the success of the mission be ensured? How will NASA ensure that the operations are safe, that they meet international and other agreements, that the timeline will be acceptable to all parties, and that scare resources will be shared equitably? A paper is being prepared which addresses these questions (Jaap & Muery, 2000).

References

Jaap, J.; Davis, E.; and Meyer, P. 1997. Using Common Graphics Paradigms Implemented in a Java Applet to Represent Complex Scheduling. In proceedings of International Workshop on Planning and Scheduling for Space Exploration and Science. 20-1—20-3. Oxnard, California. http://payloads.msfc.nasa.gov/iURC/publications

Online User's Manual for the Interim Users' Requirements Collection System: http://payloads.msfc.nasa.gov/iURC/help

Jaap, J.; and Muery, K. 2000. Putting ROSE To Work: A Proposed Application of a Request-Oriented Scheduling Engine for Space Station Operations. In proceeding of the sixth International Symposium on Space Mission Operations and Ground Data Systems (SpaceOps 2000). Toulouse France: Spacecraft Operations Oriented International Association (SpaceOps). Forthcoming.

Note: This paper is also available at http://payloads.msfc.nasa.gov/ROSE/publications