Collaborative Scheduling Using JMS in a Mixed Java and .NET Environment

Y.-F. Wang, A. Wax, R. Lam, J. Baldwin, and C. Borden
Jet Propulsion Laboratory
California Institute of Technology / NASA
Yeou-Fang.Wang@jpl.nasa.gov

Abstract

A collaborative framework/environment was prototyped to prove the feasibility of scheduling space flight missions on NASA's Deep Space Network (DSN) in a distributed fashion. In this environment, effective collaboration relies on efficient communications among all flight mission and DSN scheduling users. Therefore, messaging becomes critical to timely event notification and data synchronization. In the prototype, a rapid messaging system using Java Message Service (JMS) in a mixed Java and .NET environment is established. This scheme allows both Java and .NET applications to communicate with each other for data synchronization and schedule negotiation. The JMS approach we used is based on a centralized messaging scheme. With proper use of a high speed messaging system, all users in this collaborative framework can communicate with each other to generate a schedule collaboratively to meet DSN and projects tracking needs.

1. Introduction

NASA's Deep Space Network (DSN) provides services for spacecraft tracking and communication and for ground-based space science activities. Each project and activity has unique support requirements. Therefore, scheduling services for these activities involves multiple users with different styles of communication and types of requests. These can be simple requests for a single antenna for a specific length of time or more complex requests such as tracking Multiple Spacecraft Per Antenna (MSPA) or multiple antennas tracking one spacecraft (e.g. antenna arraying). Since the requests have a wide variety of types and there are new types frequently introduced into the system, it is somewhat difficult to recognize and adequately handle all types of requests in a centralized scheduling environment. Projects have the greatest knowledge of their tracking requirements and flexibility. Further complicating the problem, the DSN is highly over-subscribed. Therefore, a more distributed paradigm, where all projects and DSN schedulers can schedule their own tracks with their own methods in a collaborative fashion, is proposed. In a collaborative environment, all users consider their own objectives and negotiate with other schedulers to produce an overall conflict-free schedule for DSN operation. In an effort to maximize science return for the missions and to maximize antenna utilization, our proposed approach is to maintain a conflict-aware schedule that can be iterated and improved by the projects over time. Through a workflow management scheme with both synchronous and asynchronous collaborations and negotiations among affected projects, the conflicts can then be resolved.

A collaborative framework/environment was prototyped to prove the feasibility of the distributed scheduling concept for DSN resource allocation. In the prototype, Java Message Service (JMS) was used for two major modern programming frameworks, Java and .NET. In the DSN environment, there are many users on various OS platforms. Supporting both Java and Windows environments is an important step in a distributed environment. This scheme allows both Java and .NET applications to communicate with each other for data synchronization and schedule negotiation. The JMS approach we used is based on a centralized messaging scheme which is different from peer-to-peer messaging approaches such as Microsoft Indigo. Data security is one of the concerns that need to be addressed in this approach. With proper use of a high speed messaging system, all users in this collaborative framework can collaborate with each other to generate a schedule that meets both the requirements of the DSN and the tracking needs of the missions.

2. Scheduling for NASA's deep space network

The Deep Space Network (DSN) is a collection of radio antennas and their support hardware, software, and personnel. Its primary function is to connect spacecraft with their controllers at JPL and JPL's partners. Everything from calculating where to point the antennas to the work schedules of the spacecraft operators and DSN personnel starts with an official DSN schedule. Thus collaboration on the official schedule's creation is key to achieving the best results for the DSN and its users.

Figure 1: A 70-meter antenna in the NASA's Deep Space Network

The purpose of DSN scheduling is to allocate antenna times for JPL's deep space missions [1]. In addition to those missions, the DSN also supports many missions from other NASA centers as well as activities from many government agencies such as the European Space Agency, and the Japan Aerospace Exploration Agency. The DSN has ground resources distributed around the globe to handle this task. For example, The DSN has three deep-space communications facilities placed approximately 120 degrees apart around the world: at Goldstone, in California's Mojave Desert; near Madrid, Spain; and near Canberra, Australia. This strategic placement permits constant observation of spacecraft as the Earth rotates providing continuous mission coverage around the clock.

In the current process, each of the mission schedulers negotiates their mission's tracks to meet their requirements and maximize their mission's tracking time (and hence science return) for each week of the schedule. They have a number of constraints to meet. Foremost of these is the geometric constraints imposed by the positions of the spacecraft and the antennas. These constrains are called view periods. They constrain the schedule allocations to specific times. These times typically vary from day to day depending on the nature of the spacecraft's trajectory. In addition to these constraints, the schedulers must contend with limited equipment and personnel at the antenna complexes. And naturally, the mission schedulers have to contend with other schedulers who are also trying to meet their mission's requirements. Finally, there exists another class of problems that each scheduler has to deal with. If a mission is attempting to array or MSPA (or any other type of coordination involving more than one schedule item) they must meet all the above constraints as well as a new set of identification constraints so that all of the data is sent and returned in a cohesive fashion.

Not surprisingly, the complexity of schedule negotiation has been increasing steadily since the DSN's inception. The number of supported missions is constantly increasing as more missions are launched and existing missions continue to extend their tours. The requirements of the supported missions have also increased as new technology has enabled and demanded higher data rates. Advances in the equipment used to send, receive, and process data to and from spacecraft have also led to new types of conflicts between pieces of equipment on the ground. Finally, new types of communication like relaying and arraying require better coordination between missions and more accurate schedule conflict information. Due to the large cost involved, creating new resources to handle this ever-increasing complexity is not a viable option. As a result, scheduling software complexity has been increasing steadily to meet these demands.

Early schedule negotiation was done using paper and pencil. Schedules were published at fixed intervals and all of the schedulers would sit in the same room and mark up their printed copies. In this system, there was no traceability between changes and conflicts between missions were minor and easily solved by manipulating start and end times.

As complexity increased, there was a need for better visualization and statistics. This resulted in the development of a handful of scheduling applications that allowed the schedulers to view more complex relationships and try what-if analyses. Coupled with advances in electronic communication, schedulers began communicating proposals for changes to other mission schedulers. This led to further proposals for changes to other missions and helped to optimize the schedule in a very iterative manor.

As the scheduling processes evolved, the tools followed. However, as the tools matured, their limitations became apparent. The current tools allowed for a wealth of information and analysis but were not designed to be peer-to-peer communication and information sharing tools. It was still possible to share some information using shared file systems, new software technologies like SOAP, and dynamic web pages; however the schedulers still demanded a richer user experience.

The two primary groups of schedulers involved in the creation of schedules are the DSN schedulers and the mission schedulers. The former focuses on scheduling DSN-specific activities and maintaining a fair process [to all missions] that delivers schedules to the antennas in time to process all the necessary data required to track the spacecraft and send and receive information to it. The latter group, meanwhile, is focused on satisfying the requirements of a single mission (or small group of missions) and maximizing the time their missions are allotted. Thus both groups have a different idea of the "perfect scheduling system." The DSN schedulers currently use a system that emphasizes stability and keeps a record of changes. It is focused around strong processes that guarantee schedules are conflict-free with enough time for the DSN to use them to point its antennas. Naturally, this tends to a system where change is minimized and restricted as much as possible the closer the schedule gets to being used. Conversely, the mission schedulers are constantly fine-tuning the schedule to maximize their mission's coverage. As such, the environment they currently work in is very fluid and largely unconstrained. They use every form of communication available to communicate amongst themselves and the DSN to constantly improve the final schedule that is used on the antennas. The proposed environment described herein is a compromise between these two ideals. It allows for complete visibility and flexibility and facilitates communication and information sharing. It does all this while allowing fine granularity of control and complete traceability.

3. Collaborative scheduling

Each space mission has its own defined objectives and unique constraints. This domain knowledge is very specific in nature and resides within each mission. For example, each mission/project makes its own decisions on the minimum threshold in order to maintain the health of the spacecraft or mission. Also, many projects can be affected when special conditions arise such as spacecraft launch postponement and emergency operating procedures. Only mission experts can respond to such situation in a timely manner. Subsequently, there is no centralized authority that can have all the expert knowledge to dictate a global schedule to satisfy all the need. In addition, when new missions are launched, their requirements often deviate from existing requests. These new constraints are mission dependent and can best be handled at the project level.

In this environment, there are managers, project schedulers, DSN schedulers, and DSN operators. Each of them is a decision maker, information provider, and information consumer. The DSN environment is a distributed environment requiring decision making and information sharing. Each user shares the responsibility of the success of the entire network scheduling. This requires all users to work together to resolve conflicts and to come up with a schedule that achieves the overall network objectives as efficiently as possible. We call this *collaborative scheduling.*

Collaborative scheduling requires common data storage to store the *master schedule* for everyone to share as the official released schedule. This master schedule is conflict-aware. Namely, the schedule allows items in conflict to exist for negotiation. In the master schedule, all users can see the current state of the schedule and use this to help them to perform scheduling decisions. Users need to be able to play what-if games directly on top of the official schedule without being seen by other schedulers or affecting the current schedule (*dynamic workspace*). This is for real time what-if gaming. Sometimes, the user may want to share his/her what-if (draft) with a specific set of schedulers (*sharing*). For certain what-if cases, users may want to "freeze" the current schedule for an off-line case study with some other schedulers (*static workspace*). Once the study is done, the user may want to compare and consolidate the off-line schedule with the current schedule. This kind of what-if space is called *private workspace*. It contains both dynamic and static workspaces. In a more complicated case, users may want to perform various studies and make comparisons of those studies. This then becomes a *scenario management* issue.

There are two kinds of collaboration modes. One mode is *synchronous collaboration* where everyone works together at the same time to share information and perform scheduling. The other mode is *asynchronous collaboration* where users perform their operations at different time points. Synchronous collaboration can be viewed as a telephone conversation while asynchronous collaboration can be viewed as an email exchange. Collaborative scheduling must consider both of these modes separately as well as in combination with each other. For example, three missions may work together to resolve a conflict. Mission A and B may work together to come up with a proposal in real-time and then send it to mission C for consideration off-line.

Data exchange in this kind of mixed real-time operation and off-line process is a challenge. How we keep every user's local information up-to-date is the basis for true collaboration. DSN sites and schedulers are located around the world in many different time zones. Therefore, clock synchronization is the first issue to resolve. Once they all reference to the same time point, the data can then be synchronized globally. This *data synchronization* process also involves the timing issue of the data update latency.

Conflict resolution is a complicated process. It requires not only data synchronization but also process integration. When a set of schedulers come to an agreement, it may have to be sent to other groups of schedulers or to a manager for approval. The data flow within the proposal process or the approval path requires *workflow management*. The workflow management has to be a dynamic process to accommodate position changes and unexpected events.

Notification or alerting is important in a collaborative environment. Users need to be notified when data is changed or when the workflow requires their attention. Email could be used for simple human interaction with some delay. However, generally speaking, this is not an efficient medium and it could be difficult for a collaborative environment to consume. Therefore, instant or rapid messaging is more appropriate for software interfaces. Once the software receives a notification, it has to decide what to do with that notification. In data synchronization, a notification may mean the data has been changed. In workflow management, it may mean certain user attention is needed.

Collaborative scheduling requires consideration of many issues. They are generally the same issues that any collaborative environment faces. In addition, there are several issues not addressed here such as discussion forums, undo processes, traceability, change history, etc. Some of them are addressed in our prototype and some of them remain to be addressed in the future.

4. Distributed computing environment

Collaborative scheduling requires users to work together in a distributed fashion. Therefore, a distributed computing environment is needed. The objective of this distributed computing design is to connect users and resources in a transparent, open, and scalable manner.

Since resource allocation requires a collaborative effort from many users in different locations to negotiate a workable solution, the goal is to make this scheduling processing as transparent as possible, so that it operates like all mission experts are working together in the same location concentrating directly on resolving each resource issue. In addition, the methods and tools used to reach their goal should not distract them in any way. For example, the experts should not care about how requests and responses between clients and servers are required to retrieve information, as well as what network protocol is used to carry the information.

DSN Scheduling is an iterative process. Affected project schedulers will continue to negotiate on a conflict-aware schedule to resolve contentions. Openness provides them with conflict and modification awareness when it occurs. This enable projects to respond to modifications quickly. This property provides each project with a continually open environment that enables interaction with other projects until a satisfactory condition exists. If necessary, senior officials with arbitrating authority can also monitor the process and make key decisions based on task priority and urgency. As a result, openness encourages interaction to speed up the decision process.

As new missions launch and old projects retire, a scalable solution is important to address the changes in the system. The solution should be able to accommodate changes in the number of projects and resources in the DSN domain. The system should make it easy to expand and contract its resource pool to accommodate heavier or lighter loads. It should maintain its usefulness and usability, regardless of projects or resources locations.

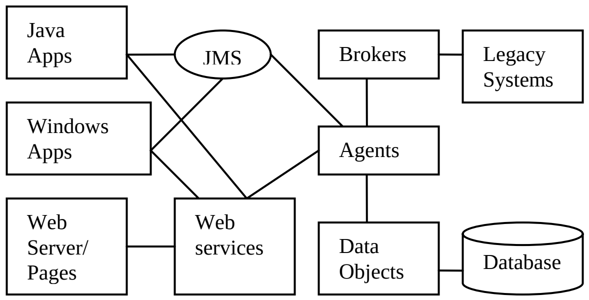


Figure 2: A simplified architecture for the distributed computing environment

To address the challenges of this distributed architecture, a multi-tier system has been designed to permit remote computing by exposing key software components as network-addressable services (see Figure 2). The system consists of the following layers:

- Client Interface layer – a front-end user interface for users to interact with the system. This interface is isolated from the business logic and database. It is also possible to quickly create new user interfaces tailored for specific uses.

- Web Service layer – an abstraction layer to access backend business logic via the SOAP messaging protocol. The service's intention is described in the XML data structures which are transported using the HTTP transport protocol to exchange information between clients and servers.

- Business logic layer – major computational intelligence and information provider. The computational intelligence is partitioned into several agents in an object-oriented fashion. This layer encompasses all scheduling related calculations, workflows, and data manipulation functions.

- Legacy Information Layer – communication element to legacy systems. This layer is used to communicate with legacy systems and provide needed information to the agents in the business logic layer.

- Data Access Layer – Data access routines to access the database. This layer transforms database information into objects that are consumed by the agents in the business logic layer.

- Data Layer – A relational database to store schedules, constraints, ownerships, workflow, etc.

- Messaging layer – a messaging server to provide messages and alert functionality for the system.

In modern network computing environment, Java and .Net frameworks are the major players. Their market shares are expected to be 50-50 in year 2006. Java dominates the large-scale enterprise market due to its multi-platform nature and adaptation to major database/solution venders such as Oracle, Sun, and IBM. .NET dominates the small and mid-size business market due to its initial cost and performance benefits. We don't expect either platform to dominate this domain in the next few years. Therefore, we should expect them to co-exist and we should design our system to allow both of them to contribute their strengths. Market share is one of the metrics used to justify decisions over which platform to adopt in a project. There are major strengths for both Java and .NET development. Java focuses on multi-platform solutions while .NET distinguishes itself by embracing multiple language development. As both .NET and Java embrace Web services, cross-platform communication is becoming less of an issue. This is strengthening the trend toward mixed .NET and Java environments. Therefore, our strategy is to take advantage of both frameworks and utilize them wherever technologies can best fit to our needs.

Our current prototype is designed with the above philosophy in mind. We take advantage of the existing investment and PC user base by developing on the .NET platform to provide rapid deployment of this proof-of-concept prototype. For the multi-platform client and overall messaging, we adopt Java-based JMS to enable rapid messaging and data synchronization. With the popularity of SOAP and XML, the system also adopts these technologies for exchange of information in our distributed environment.

5. Messaging and data security

One of the main goals of our effort is to introduce real-time collaboration into the scheduling process. This task is complicated by the fact that the participants in the scheduling process are both distributed across different locations and using different platforms. Developing multiple sets of code for multiple platforms to coordinate in real-time is very impractical.

In this prototype, the middle-tier agents were written in C# using the .NET framework to take advantage of existing systems. However, we chose to use the Java Message Service (JMS) [2] for messaging due to its maturity and ease of use. The issue of how the C# end of the service would employ JMS was solved by using IKVM[1]. IKVM.NET is an implementation of Java for Mono [3] and the Microsoft .NET Framework. It includes the following the components: a Java Virtual Machine implemented in .NET, and a .NET implementation of the Java class libraries and tools that enable Java and .NET interoperability.

IKVM was used to convert the core JMS .jar files containing the needed Java classes to the equivalent C# 'binary' files that run with .NET and the IKVM provided library files. It was necessary to point the Microsoft development tools at these libraries and write code that, for all intents and purposes, looked like java code but in C#. This fact was a benefit as sample code to implement the messaging functions written in java was easily adapted to C# as the names of the methods, constants and classes were identical.

While collaboration was designed to be real time, the fact is that when someone is active in say, Spain, someone is sleeping in Australia. When a person starts their work session they need to be informed of any changes that may have occurred since they were last active. JMS provides an excellent solution to the problem with the use of persistent messages.

JMS messaging has two basic paradigms. One is a point-to-point message. The creator of the message sends the message to a particular receiver and if they are not there, the message is lost. This type of messaging is handled by something called a *queue*.

Persistent messaging is a broadcast paradigm where one creator sends a message and multiple people receive it. This is termed a *topic*. Persistent messages have guaranteed delivery while messages in a queue do not. If message delivery fails, the message persists until it is safely read or the lifetime of the message expires. The creator can specify how long the message lives before it is deleted from the system. In our work we used persistent messaging throughout.

Messages in JMS are 'fire and forget'. This means that if the creator of a message successfully delivers the message to the JMS server, the creator need take not any more action to guarantee delivery. The server takes care of the delivery process and coordination with all the intended receivers.

[1] See http://www.ikvm.net/#Introduction

Messages may be read asynchronously or synchronously (polled). Since it was simpler to be notified when a message had arrived, and have the message in hand simultaneously, that approach was used. However, since messages are persistent and dispersed in time, a client that wishes to control when messages arrive, due to some critical activity for example, has the choice to so.

It is important to note that what travels between the server and the clients listening for changes are not the changes but a notice of a change. It is up to the client to request whatever changes are needed through the data channels. Namely, JMS remains as a message services rather than a data service. This has benefits of performance and security. Performance-wise, this cuts down on traffic between the server and the client. It is also worth noting that the volume of messages traveling around is small. A single JMS server can handle all the traffic as there are no more than dozens of messages per minute normally. Security-wise, this only delivers notification without the data itself. Once a JMS message is received clients determine the activity needed and go through regular data channels to exchange data in a secured way.

Security in the messaging service itself was addressed in two simple ways for the messaging system. The first was simply the use of a firewall. Only those users behind the company firewall could see the JMS server and initiate activities with it. Users also need user ids and passwords to access the server even behind the firewall. Only certain users are privileged to see scheduling activity information. What was not addressed directly by the project was dealing with attacks over the net. It is possible for someone to take over a privileged user's machine and send a flood of messages perhaps crippling the JMS server. They may also simply watch the traffic going by. We deemed the network security provided by the administrators of the company networking environment should be sufficient.

6. The prototype

A collaborative environment for DSN scheduling was prototyped to prove the concept of collaborative scheduling. In this environment, all users refer to a single master schedule while they can do collaborative what-if gaming with a subset of the users in real-time on top of the master schedule. This is the concept of dynamic workspaces where any change to the master schedule can be seen immediately by all users while each individual user may have their own draft on top of the master schedule. This draft can only be seen by selected users assigned by the draft owner. Users can also copy the master schedule to a static workspace. Then collaborative scheduling can take place off-line in the static private workspace within a subset of the users. Once the negotiation or preparation is done, users can compare the off-line schedule with the on-line schedule. Any change to the master schedule has to go though the workflow management system to guarantee proper ownership is observed. All changes that go through the workflow process become items in the master schedule. The concept of ownership is also prototyped in this system to identify the owner of a facility or mission in certain timeframes. This is an integral part of the workflow management. The system also features a real-time conflict checker. The real-time conflict checker tells users where the conflicts are and a rescheduling engine can be called remotely to calculate possible resolutions. A reporting system is also built to report scheduling related metrics such as tracking time and supportability information.

With the need for real-time data update for each user and event notification for both users and software, a messaging system was built. This messaging system includes automated email notification for certain events and a rapid messaging system of instant alerts to software (and then to users). Java Messaging Services (JMS) was used to implement the rapid messaging system. JMS was used to deliver messages to the message bus while all actual data was handled through the data bus. This makes the system architecture clear and data secure. In consideration of the multi-platform distributed environment, the prototype system allows JMS to work on both Java and .NET frameworks. This enhances the true distributed computing concept. With clients implemented in various frameworks, our JMS approach allows them to work together in a mixed environment using one messaging service and then exchanging data using SOAP/XML standards.

In this prototype effort, we extended the current DSN scheduling middle-tier layer to host the computation engine. This includes schedule conflict checking, data synchronization, dynamic workflow determination, etc. A database is established to host the master schedule, dynamic workspace, static workspace, workflow management, and traceability/change history. Three client applications were built to demonstrate the flexibility of the system.

A Windows client (see Figure 3) was built to show the concept of an Integrated Analysis Environment (IAE) where every user can perform all analysis work within the same environment. The analysis environment is flexible and has a lot of analysis tools (in sub-windows) so users can arrange their own working environment. Property grid is also integrated into the environment so the property of any selected object can be easily displayed and modified (if that specific property can be modified by the current log-on user). The tree view allows users to see data in a more hierarchical way. The calendar view allows users to see data in a more familiar appointment book type of view. The graphics shows a schedule Gantt chart and a supportability bar chart. The list view allows users to see and modify data in a more linear fashion. The application console displays events and messages in a multi-color scroll view to inform users of the current system status. All analysis results are kept up-to-date among all views. Namely, all analysis data such as supportability and conflict information are synchronized with the current schedule and recomputed if the schedule changes. This is based on the event model we used in the prototype. The master schedule is synchronized constantly among all users who use this environment without requiring any interaction from the user. This is based on the messaging model in the prototype. All private workspace items are marked clearly on top of the master schedule. The environment also screens the applicable JMS messages to report in the application console.

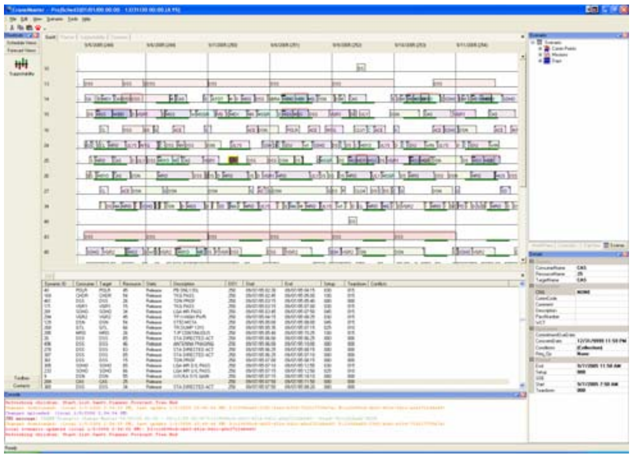


Figure 3: An Integrated Analysis Environment in Windows

A Java client (See Figure 4) was built to show multi-platform access to the same back-end and to demonstrate a way to handle delayed updates. All changes to the master schedule are marked in different color in the Gantt chart. Users can decide when to update the local schedule. This allows users to work on their draft without perturbation to the schedule they are currently working on.

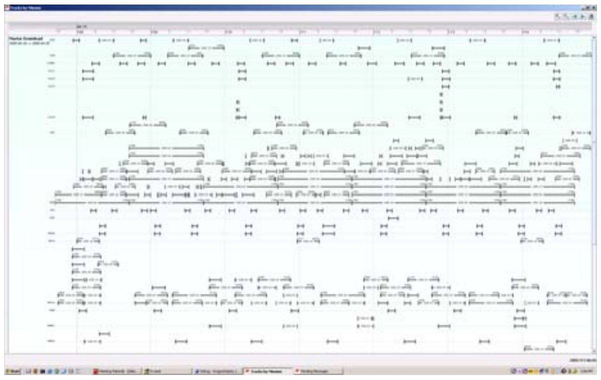
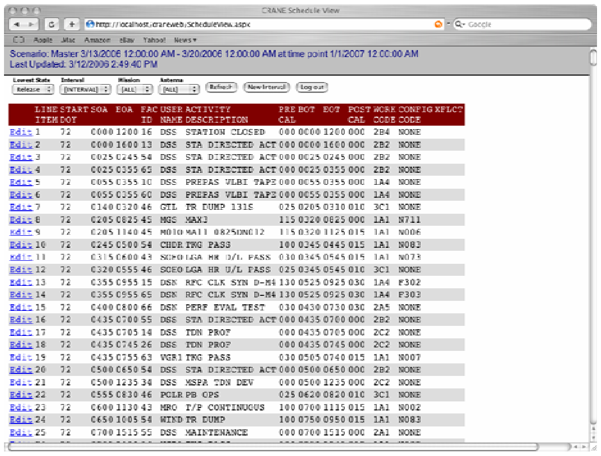


Figure 4: Java application

A Web application (see Figure 5) was also built to demonstrate the feasibility of web page access. The current web page prototype only allows data viewing. However, this can be easily extended to allow for data input. This application can be run in any browser on any platform. The example figure was captured using Safari on a Macintosh.


CRANE Schedule View

http://localhost/craneweb/ScheduleView.aspx

Scenario: Master 3/13/2006 12:00:00 AM - 3/20/2006 12:00:00 AM at time point 1/1/2007 12:00:00 AM
Last Updated: 3/12/2006 2:49:40 PM

Lowest State: Release; Interval: [INTERVAL]; Mission: [ALL]; Antenna: [ALL]

	LINE ITEM	START DOY	SOA	EOA	FAC ID	USER NAME	ACTIVITY DESCRIPTION	PRE CAL	BOT	EOT	POST CAL	WORK CODE	CONFIG CODE	XFLCT
Edit	1	72	0000	1200	16	DSS	STATION CLOSED	000	0000	1200	000	2B4	NONE	
Edit	2	72	0000	1600	13	DSS	STA DIRECTED ACT	000	0000	1600	000	2B2	NONE	
Edit	3	72	0025	0245	54	DSS	STA DIRECTED ACT	000	0025	0245	000	2B2	NONE	
Edit	4	72	0025	0355	65	DSS	STA DIRECTED ACT	000	0025	0355	000	2B2	NONE	
Edit	5	72	0055	0355	10	DSS	PREPAS VLBI TAPE	000	0055	0355	000	1A4	NONE	
Edit	6	72	0055	0355	60	DSS	PREPAS VLBI TAPE	000	0055	0355	000	1A4	NONE	
Edit	7	72	0140	0320	46	GTL	TR DUMP 131S	025	0205	0310	010	3C1	NONE	
Edit	8	72	0205	0825	45	MGS	MAX3	115	0320	0825	000	1A1	N711	
Edit	9	72	0205	1140	45	M01O	MAI1 0825DN012	115	0320	1125	015	1A1	N006	
Edit	10	72	0245	0500	54	CHDR	TRG PASS	100	0345	0445	015	1A1	N083	
Edit	11	72	0315	0600	43	SOHO	LGA HR D/L PASS	030	0345	0545	015	1A1	N073	
Edit	12	72	0320	0555	46	SOHO	LGA HR U/L PASS	025	0345	0545	010	3C1	NONE	
Edit	13	72	0355	0955	15	DSN	RFC CLK SYN D-M4	130	0525	0925	030	1A4	F302	
Edit	14	72	0355	0955	65	DSN	RFC CLK SYN D-M4	130	0525	0925	030	1A4	F303	
Edit	15	72	0400	0800	66	DSN	PERF EVAL TEST	030	0430	0730	030	2A5	NONE	
Edit	16	72	0435	0700	55	DSS	STA DIRECTED ACT	000	0435	0700	000	2B2	NONE	
Edit	17	72	0435	0705	14	DSS	TDN PROF	000	0435	0705	000	2C2	NONE	
Edit	18	72	0435	0745	26	DSS	TDN PROF	000	0435	0745	000	2C2	NONE	
Edit	19	72	0435	0755	63	VGR1	TRG PASS	030	0505	0740	015	1A1	N007	
Edit	20	72	0500	0650	54	DSS	STA DIRECTED ACT	000	0500	0650	000	2B2	NONE	
Edit	21	72	0500	1235	34	DSS	MSPA TDN DEV	000	0500	1235	000	2C2	NONE	
Edit	22	72	0555	0830	46	POLR	PB OPS	025	0620	0820	010	3C1	NONE	
Edit	23	72	0600	1130	43	MRO	T/P CONTINUOUS	100	0700	1115	015	1A1	N002	
Edit	24	72	0650	1005	54	WIND	TR DUMP	100	0750	0950	015	1A1	N083	
Edit	25	72	0700	1515	55	DSS	MAINTENANCE	000	0700	1515	000	2A1	NONE	

Figure 5: Web application

The prototype has been shown to some DSN schedulers, project schedulers, and managers and positive feedback was received. Several JPL software projects are currently looking into the use of JMS. This prototype provides an example of how JMS can be used in a mixed-framework environment.

7. Conclusion

Collaborative scheduling is needed in a scheduling environment where all the needed information and decision making is not controlled by a single source. When efficiency is needed, it is better to allow all resource consumers to schedule their tasks collaboratively. A DSN collaborative scheduling system was prototyped to prove this concept.

In this prototype, a rapid messaging system using Java Message Service (JMS) in a mixed Java and .NET environment was established. This scheme allowed both Java and .NET applications to communicate with each other for data synchronization and schedule negotiation. Combined with the schedule data structure, ownership, and workflow management, this system allowed both synchronous and asynchronous collaboration in both dynamic and static workspaces with full conflict and traceability information.

8. References

[1] C. Borden, Y.-F Wang, G. Fox, “Planning and Scheduling User Services for NASA’s Deep Space Network,” *NASA Planning and Scheduling Workshop*, 1997.

[2] Richard Monson-Haefel, David Chappell, *Java Message Service*, O'Reilly Media, December 2000.

[3] Edd Dumbill and Niel M. Bornstein, *Mono: A Developer's Notebook*, O'Reilly Media, July 2004.

Acknowledgement

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.