Design of a Deep Space Network Scheduling Application

Bradley J. Clement and Mark D. Johnston

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
{bclement, mdj}@jpl.nasa.gov

Abstract

The Deep Space Network (DSN) is a collection of ground antennas used to provide communication services for space missions at and beyond Earth. Currently, nearly 30 people work full time to schedule 61 missions over a ten year horizon using a few disconnected tools. The large manual and collaborative effort to generate schedules and resolve conflicts has spurred an effort to build a new scheduling application incorporating automated scheduling. We will describe the basic design of this application, the Service Scheduling Subsystem (SSS), and discuss some of the design issues. We describe the messaging interface to the automated scheduler, including the request and constraint language and support for user-defined constraints. There are many other features that we believe will heavily influence the success of the application. These include an efficient schedule editing interface, request management, non-interfering suggestions from automated scheduling, user-defined alerts, hypothetical schedules, and automated negotiation.

Introduction

The Deep Space Network (DSN) consists of three complexes of ground antennas located in California, Madrid, and Canberra. There are 16 antennas that range in size from 26 to 70 meters in diameter. This network primarily supports interplanetary missions and radio and radar astronomy but also supports some Earth orbiting missions. DSN services for spacecraft include command uplink, data downlink, tracking, and navigation. While 150 missions are listed as DSN users, about 20 spacecraft are allocated resources in a 4-month schedule.

Schedules are currently manually generated a year into the future with allocations to the minute. These are currently generated a week at a time and average 370 tracks (allocation of an antenna to a mission over a time period) per week. These tracks are typically 1 to 8 hours long and must be allocated in a viewperiod (i.e. a time period when the spacecraft is visible to the antenna). There are around 1650 of these viewperiods defined per week. Clement and Johnston describe the scheduling problem in greater detail (2005).

The greatest inefficiency of the current scheduling process is not in generating low-conflict schedules but in trying to meet the ill-specified requests of the missions. (The DSN calls these requests "requirements," but we will call them "requests" to avoid confusion with software requirements.) The requests used for generating schedules are currently only specified in hours/week per antenna. DSN schedulers need more information to determine the acceptable duration and frequency of the tracks generated. While missions convey some of this information over the phone or e-mail to schedulers, there are no records of this additional request information, and subsequent proposed schedule changes often conflict with the intended requests. These changes often affect multiple missions, resulting in frequent meetings, e-mails, and phone calls to rework the schedule.

Historically, scheduling for the DSN has been split into three horizons (0 to 8 weeks, 8 weeks to 2 years, and beyond 2 years) with more detail and conflict checking for nearer-term schedules. The missions have scheduling representatives who work with other groups of schedulers who can update the database. Each group has their own scheduling tools with almost no overlap. Currently, there are around 30 full time employees scheduling for the DSN.

A working group at the Jet Propulsion Laboratory (JPL) developed a set of design principles for an improved DSN Scheduling process and system:

- Seamless scheduling for all planning horizons
- Remove artificial temporal boundaries
- A master schedule always exists, visible to all users
- A single process and tool suite is used by all classes of users
- Requests and schedules are fully traceable
- Conflicts are resolved at the lowest level possible, peer-to-peer by default
- Meetings are called only as needed (e.g., to resolve difficult conflicts or to address an emerging asset contention period)

- Workspaces are provided to users to develop requests (especially for what-if analysis)
- Need both web-based and workstation-based capabilities
- Distinguish global (shared) workspace and local (private) workspace; private workspace may span a set of peers
- Need scalability (loading, # users, # assets) and extensibility (evolving technology)

The most dramatic change to the process will be to give missions direct control of the schedule. Instead of having to help another group of schedulers understand their requests and then check the resulting schedule, the missions will be able to directly change the tracks in the schedule for their mission.

Based on the recommendations of the working group, JPL has begun development of a Service Scheduling Subsystem (SSS). The users of this system would be the current schedulers (who work out the logistics of scheduling), mission operations staff (who provide the objectives/requests of scheduling), managers (who look at resource availability and perform what-if analysis for different mission scenarios), and antenna complex maintenance staff (who must ensure that the antennas serviced regularly to maintain operability). The basis of the design of the SSS is a set of functional requirements provided by a representative group of users.

Service Scheduling Subsystem

The major functions of the SSS are a graphical user interface (GUI), data management, automated scheduling, and reporting, as shown in Figure 1. We describe some of the design issues related to these functions, giving special attention to an automated scheduling service, called the Scheduling Engine (SE). The extension of the request language is reported elsewhere (Clement & Johnston, 2005).

The functions provided in the subsystem as depicted in Figure 2-1 include a:

- User Interface
  The User Interface function provides a user with the interactive graphic displays and text input and output mechanisms needed to enter all relevant scheduling items and to display them as well as the schedules generated.
- Database Management
  There may be more than one database with this subsystem, but there shall be only one master schedule. In addition to the master schedule, a record of requirements transactions or change history, along with the chronology of requirements input for traceability (of scheduled activities to original requirements), management of private workspaces, and contingency schedule support is contained in this function.
- Schedule Generation
  The automatic Schedule Generation function consists of several elements necessary for maintaining the master schedule and Private Workspace schedules. These elements include retrieving up-to-date view periods and equipment status. The heart of the scheduling function is a scheduling engine that generates schedules, finds conflicts, and then reduces the number of conflicts in the schedule.
- Reporting
  The Reporting function provides network load forecasts for supporting 'what-if' studies and other reporting features for the subsystem including user defined reports, conflict reports, metrics, view period files for sequencing, allocation files for sequencing, and traceability reports.

The data flow among these major components is shown in Figure 2. Some amount of functionality will be executed on the user's computer for the user interface. Part of this involves administration to manage user permissions, to update the list of assets and when they are available, and to modify constraint rules describing legal allocation of resources. For example, two spacecraft may use the same antenna if only one is uplinking and if the antenna can point to both spacecraft at the same time (e.g. both at Mars). These kinds of rules change over the years when new service capabilities or new kinds of missions come along. The Service Preparation Subsystem (SPS) calculates and stores viewperiod information based on trajectory. These calculations take into account many physical properties of antennas, including cable wrap and keyhole constraints. They are calculated independently from the schedule by always "stowing" the antennas back to a neutral position after every track.

Scheduling Engine

The intended use of automated scheduling is to provide suggestions for schedule changes based on request changes. This also includes generating a schedule from scratch based on bulk input of requests. Automated scheduling is made available to the rest of SSS as a network service using messaging.

User interaction with the SE is session-oriented. To begin any interaction, the user initiates a session which persists until the interaction is complete, or until the session times out and is destroyed by the SE. Interactions that refer to non-existent sessions will return an error status.

Session initiation incorporates an authentication and authorization process, which makes use of a standard authentication and authorization mechanism used elsewhere in SSS. For the purposes of this interface,

sufficient information must be provided in the interface to enable appropriate authentication and authorization of the user.

Initiate Session Request – initiates a session for scheduling and includes information about the user, the purpose of session, and timeout conditions.

Initiate Session Response – the status of the session as a response to an Initiate Session Request message, which includes a session ID among other information described below. All request messages are processed in a session identified by ID.

Terminate Session Request – terminates a specified session, possibly in the middle of processing other messages.

Status Request – requests status of a specific session

Status Response – the status which includes whether a session exists, is "ok," has an error, and/or is busy; what fraction of processing has completed; a description of any error; the session ID; an ID for the current or last completed processing step (to use as a savepoint); and the timeout conditions

Generate Schedule Request – generates a schedule of tracks with instructions on what requests/requirements to schedule, the time range within which to schedule, what schedule and viewperiod data to take into account (e.g. missions, assets, workspace), whether to return the entire schedule or just differences, a time limit for processing, scheduling strategies to use, any changes to default constraint rules

Generate Schedule Response – the generated schedule, the changes made to the schedule, or both

Identify Conflicts Request – requests conflicts with constraint rules and/or requests in a specified schedule (time range, which missions, assets, workspace) and within a specified processing time limit

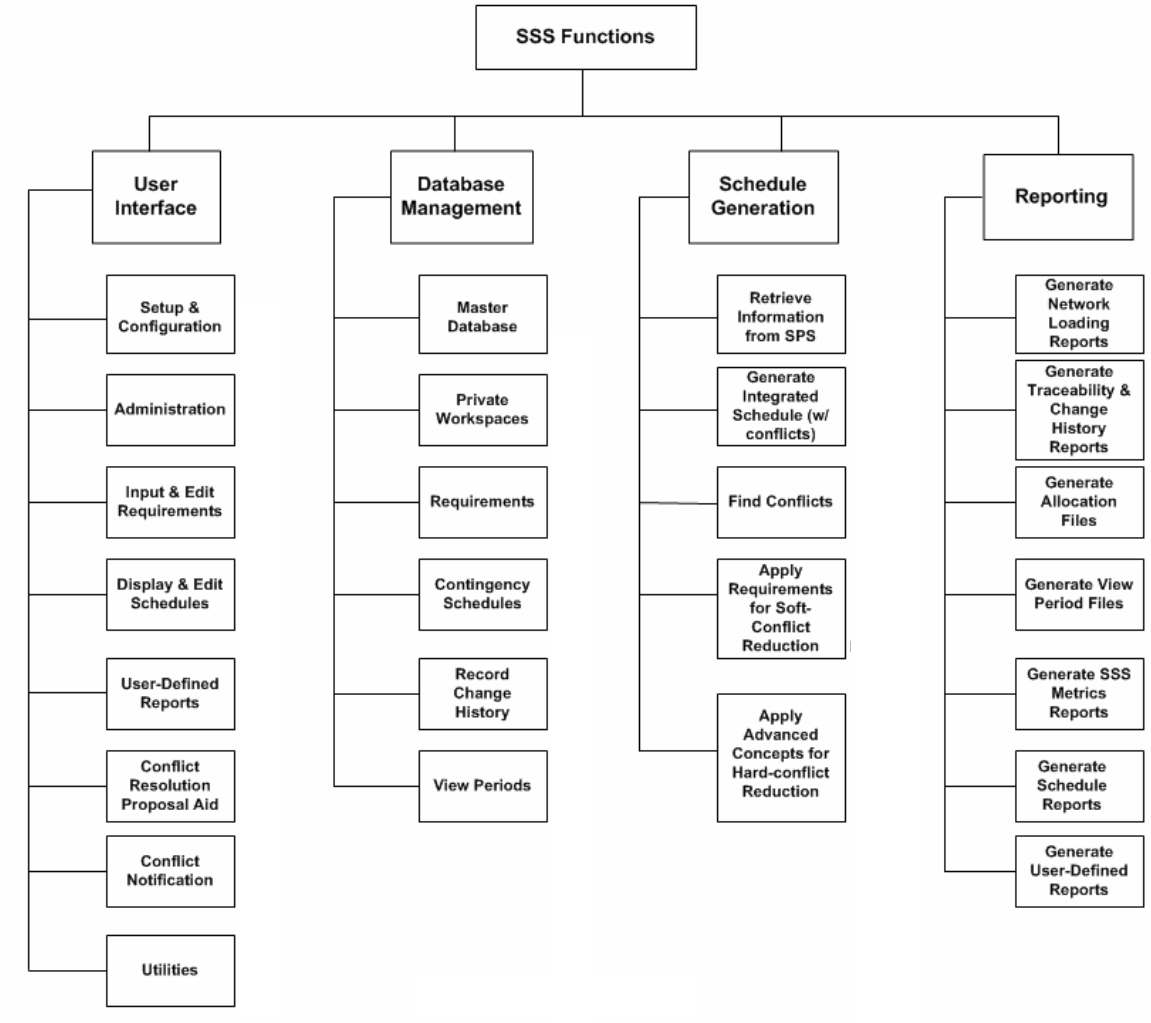

Figure 1: Service Scheduling Subsystem Description

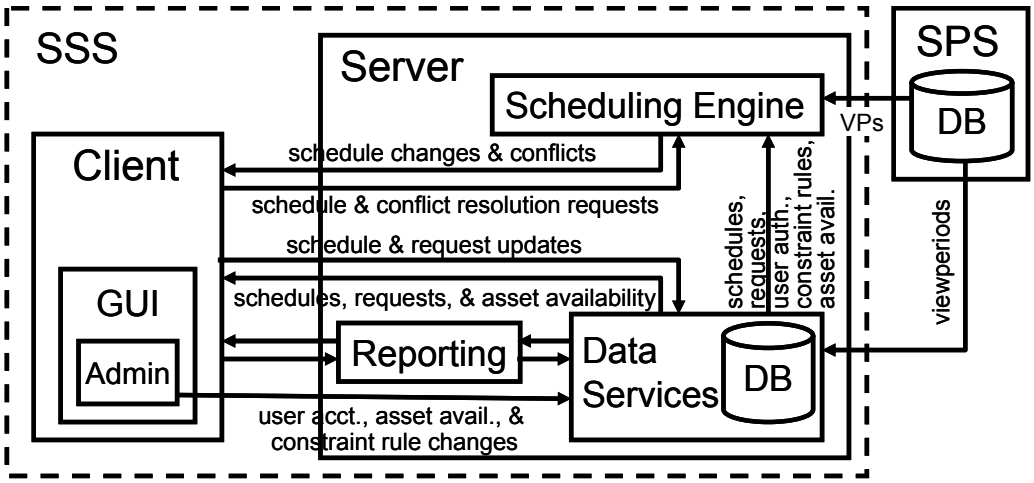

Figure 2. Data flow among components of SSS and the SPS database

Identify Conflicts Response – the status and a set of conflicts. A conflict includes a type, attributes, and associated tracks.

Resolve Conflicts Request – this is the same as the Generate Schedule Request message except there are no request/requirements to schedule.

Resolve Conflicts Response – this has the same structure as the Generate Schedule Response message.

Apply Schedule Changes Request – schedule changes to apply to a session, possibly in the middle of processing other messages

Apply Schedule Changes Response – the session status in response to applying schedule changes

Terminate Processing Request – terminates processing of any prior request messages

Undo To Step Request – reverts the schedule back to its state when status was reported for the given step ID.

Undo To Step Response – the session status in response to an Undo To Step Request message

Figure 3 illustrates an example of interactions that occur when a user generates a schedule. The participants in this diagram include:

1) SSS scheduler user: this represents the customer for the schedule, working through other components of SSS (e.g. client GUI or web-based interface). (Note that the end user does not directly interact with the SE.)
2) SE manager: this is the server process that interacts with the user to handle the generate schedule request, and with internal elements of the schedule engine to satisfy the request. The SE manager is always online to handle user interactions.
3) SE session: a transient process or set of processes that perform the computational and database actions required to satisfy a user request, or a related series of requests. Because a session maintains the state of a particular interaction with a user, incremental interactions are readily supported without long delays for re-initialization. Many sessions may exist at one time within the SE, to handle many simultaneous user interactions. They may be distributed across multiple hardware components for performance and load balancing.
4) SSS DB: the SSS database, from which schedule and related data is retrieved, and to which schedule changes and status data is stored. The SE session may also connect to the SPS database, for viewperiod and asset status/schedule data, but that is not shown in this diagram.

This message interface to the SE enables a wide variety of ways to schedule on behalf of a user in terms of who (or what) submits messages and when they are submitted. One approach briefly considered was to not allow the users to make edits to the schedule and have the SE schedule all requirements and resolve all conflicts. Thus, the users would manage their schedules by editing their requirements. Instead, we decided that the SE should only offer suggestions and that only the user could commit suggested changes to the database. Because conflict resolution is the responsibility of the users, missions will want to schedule their requirements as early as possible in order to resolve conflicts as early as possible. While there is no formal prioritization of missions or their requests, it is usually common sense which mission will win a fight for a resource. For example, launches and orbit insertion take priority over telemetry, and primary missions have precedence over extended missions. Thus, in a collegial atmosphere, there

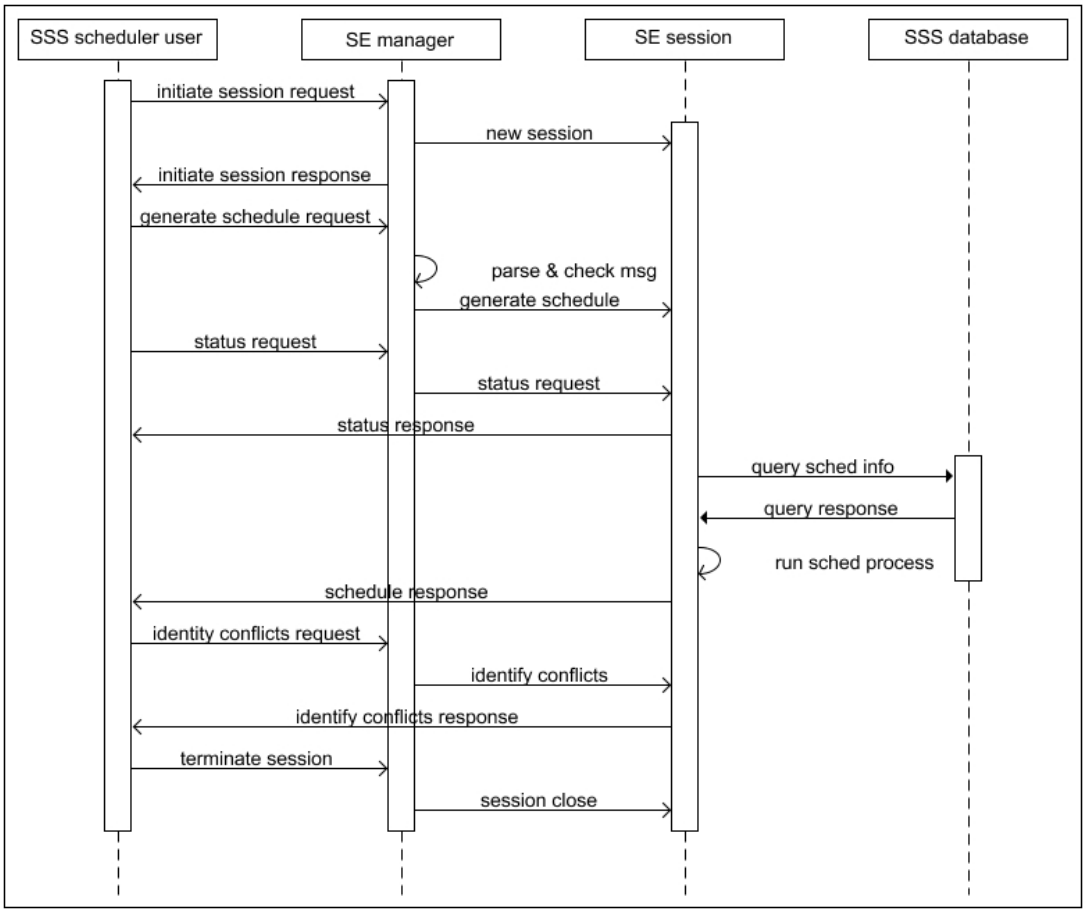

Figure 3: UML sequence diagram of interactions for the "generate schedule" scenario

is no reason to require all missions to supply their requests at the same time for a third party to schedule.

Another simple approach to using the SE interface would be to have a webpage where the user fills out the fields of different messages and the SE processes them. However, it is more beneficial to predict what the user might want to process and make unobtrusive suggestions. For example, if a user works only in a few private workspaces, once the user log into SSS, SSS can invoke the SE to resolve any conflicts in those workspaces. This could even be done when the user is offline and a schedule change affects a private workspace. If the user is drafting new requirements, there is no point in waiting for the user to push a button to invoke the SE. The SE could be hypothetically scheduling the requests (in its own private workspace) as the user drafts. This can be very useful in visualizing the requirement as a potential schedule and indicating to the user mistakes where the specified request is not what is intended. Having solutions ready when the user wants them could be the difference in whether a system like SSS is successful.

Another intended use of the SE interface is to allow users to "bend the rules" in order to more flexibly explore hypothetical changes by selectively choosing what schedule and constraint data to schedule against. This is especially useful in determining whether to change requirements to resolve a conflict. For example, a user may have hardened a requirement to lock down on a particular time and antenna so that the mission could assume that the schedule was fixed in order to continue operations planning. However, if another mission requested the same resources for a more critical activity, then the user could easily disable the request to lock down on the time and resource and explore possible changes. Users can also override user authorization constraints on what mission's data they can modify in order to explore a ways other missions can resolve the conflict. Then, the workspace can be sent as a proposal to the affected missions to get their approval for the changes. A future work item is to schedule based on multiple objectives so that users can propose schedule changes that benefit multiple missions or minimize the concessions needed to resolve a conflict.

A significant challenge for the SE is being able to schedule for user-definable constraints. If constraint rules for antennas and missions change over the years, it will be costly to make code changes to be able to schedule for the new rules. Thus, it is important that the SE (in effect) be able to schedule for unknown problems. We take a somewhat passive approach by allowing constraints to be specified as SQL queries and taking general approaches to rescheduling that we hope to be effective. The idea is to recognize conflicts as SQL queries that return records with track identifiers. Then, those tracks are considered to be involved in the conflict(s) and are possibly causing the conflict(s), so they will be considered for rescheduling. A local search approach is effective at rescheduling with minimal changes. Both local search and exhaustive search algorithms can be used in conjunction to reschedule with minimal changes (Clement & Johnston, 2005).

Data Management

There are a few subtle issues in managing workspace, request, and change history data and implementing an undo mechanism. Users wanted change history in order to analyze how a mission's requests have changed over time (to see if the mission is asking for more resources than initially agreed to by NASA, for instance). Undo is simply used to back out of changes while editing. Undo information is basically the same as detailed change history for a short period time, so there is no need to have separate representations and storage for these similar concepts.

One basic question to answer is how will schedule data be committed to the master schedule or to a workspace. One choice is SSS automatically commits every detailed edit to the database, and the undo mechanism can reverse edits made in prior SSS sessions. Another approach is to commit changes only when the user explicitly saves the edits, in which case undo/change history information could be stored up until the last save, and prior saves could be change history checkpoints. Then, for how long should undo/change history information be saved? Our current thought is that without an explicit save mechanism, unintended changes could more easily be committed without notice, so we lean towards having an explicit save. Undo/history information could be kept at larger granularities for older data. For example, checkpoints could be saved once a week for changes made more than 6 months ago.

Now what if two users have permissions to the same mission, and one wishes to undo their committed edits even though the other user committed edits concurrently. Should the undo not be allowed? Should it only be allowed if also undoing the edits of the other user? Should no more than one user have permissions to the same mission (or same mission and timeframe)? If the users' edits are in non-overlapping timeframes then it is possible to undo the edits of one without the other, but if only the tracks edited are different, it is not obvious how to determine separability since one user's edit could be indirectly caused by the other's even if they involve different tracks. While we wish to be able to smartly detect whether user changes are separable in SSS, this may require a lot of development, so we aim to first not bother with separating edits and to request confirmation from users before allowing them to undo other users' edits.

Workspaces are schedules derived from the master schedule or another workspace, and they can easily become inconsistent. For example, if workspace B is derived from workspace A, and some user attempts to delete a track in A that B has modified, what is the appropriate behavior?

1. The track is not allowed to be deleted in A.
2. The track is deleted only in A and committing B (after a warning) will add the track back to A.
3. The track is deleted in A and in B (and users of B are warned).
4. The track is deleted in A, and users accessing B are notified of the deletion and must resolve the inconsistency before committing any more changes (or the possible resolution choices are listed for the user select).

We believe the fourth option is the safest, but the third option is much easier to implement while still being a reasonable alternative.

The latest version of Oracle (10g) has a workspace management feature that allows the database to manage a workspace data as changes to another workspace (or the master). It appears that the SSS workspaces can be easily implemented using OWM (Oracle Workspace Management). Change history checkpoints are also just changes relative to other data, so it is also possible that change history and even an undo mechanism could be implemented in OWM.

Managing request data is another challenge for SSS. Users want to know what requests have been satisfied, what have not, and how that has changed over time. The database needs to maintain a link between a request and the tracks in the schedule that fulfill it (or try to fulfill it). Users also want to edit the schedule directly and independently of the requirements. A problem arises when a user's edits conflict with the associated request, but the user wants the edits. In the past the integrity of requests was enforced such that edits were not allowed to violate them, but the users explicitly asked to break the link between the requests and the schedule because it was too annoying to edit the requests in order to edit the schedule. SSS is required to maintain this link, so either the schedule is allowed to disagree with the requests (and the user is not pestered with conflict messages), or the

requests must be automatically repaired to agree with the schedule edits. A simple way to repair a request is by using an override request where one requirement supersedes or replaces the other. The overriding request can simply be to have the track as edited, and it overrides the prior violated request. The disadvantage to adding overriding requests is that many edits can lead to a bloating of requests such that they are difficult to manage. We find that either of these solutions is acceptable and hope to provide both capabilities in SSS.

User Interface

The user interface design is arguably the most important part of SSS since it is difficult to create intuitive, efficient user interfaces, and there are no greatly successful mixed-initiative planning applications to serve as a model for SSS. Creating a user interface for editing temporal constraints in MAPGEN for the Mars Exploration Rover mission was the key to its adoption, but editing temporal constraints is just one aspect of creating requirements (Bresina et al., 2005). PASSAT (Plan-Authoring System based on Sketches, Advice, and Templates) is an ambitious vision for interactive tools for general mixed-initiative planning (Myers et al., 2003) and builds on a prior mixed-initiative system (SIPE-2) that has had success in military domains (Wilkins, 1993). We try to follow some of their design principles. By providing suggested schedule changes to meet higher level requests, SSS helps users fill out sketches of plans. The automation of scheduling is completely controllable by the user since the SE only provides suggestions, and only the user can commit changes to the database. SSS is flexible like PASSAT in its ability to allow users (in a workspace) to turn constraint rules on and off, to lock and unlock track timing and asset allocations, to modify tracks of other missions, to override requests when making inconsistent edits to the schedule, and to choose which parts of the schedule (missions and assets) to consider when scheduling. We increase usability like PASSAT by allowing checkpoints to be saved and planning operations to be undone. While PASSAT shares many of the same user interface objectives as SSS, there are other notable mixed-initiative planning systems that Myers et al. briefly describe as related work (2003).

As mentioned earlier, users can send or share private workspaces to propose changes a group of missions can make to resolve a conflict. Using an automated scheduler to recommend solutions can greatly reduce human effort in finding resolutions to conflicts, but it does not relieve the communicative part of the negotiation process. One approach is to use an automated multi-objective scheduler to try to find solutions that favor all missions involved. This can reduce the number of rejections and counterproposals in the negotiation. Another complementary approach is for the mission to allow SSS to generate proposals and responses automatically. While the user would not want the system to make all decisions, in some cases the decision is obvious. For example, the system should always refuse proposals to reschedule a critical track and should always accept proposals that fulfill unmet requirements. For proposals to which the system may not automatically respond, it can make suggestions, including counterproposals (based on multi-objective scheduling). As users gain more confidence, they can define situations in which the system is allowed to respond automatically. Not only can this reduce the human involvement in negotiation, but also more negotiations can take place in order to create better overall schedules. This same idea applies to a single user scheduling for a single mission—the user could authorize SSS to make scheduling decisions without user-intervention under certain conditions. These conditions could be specified as SQL and processed similar to user-defined constraint rules.

Other required features of SSS are user-definable notifications (such as e-mail when a change conflicts with a user's mission) and user-definable reports. Both of these are similar to the user-definable constraints and user-definable automated response conditions. The underlying language can be SQL. A challenge is how to get users to express SQL through a GUI without having them learn SQL. As this is a common database usability issue, existing tools may work suitably. The user-defined reports have an extra layer of complication in how to get users to express a graphical layout of data return from SQL queries. Again, there are existing approaches that may be fruitful (e.g. cascading style sheets).

There are many other user interface issues that we have yet to work out. One particularly important aspect of the user interface is how the user will interact with the scheduling engine. In the Scheduling Engine section, we discussed ways that the SE could be invoked efficiently, but the manner in which suggestions are presented to the user is equally important. Another is drag-and-drop editing of tracks—how can a user use a mouse to place a track to start on a specific minute in a Gantt chart displaying a week schedule?

Other Related Work

In general, space applications often differ from others in that scheduling requires rich languages for modeling temporal relationships, spacecraft instruments, and dynamics (e.g. Eggemeyer et al. 1990, Muscettola et al. 1998, Chien et al. 2004, and Ai-Chang et al. 2004). For DSN scheduling, the complexity of antenna resources is removed by restricting how they can be used. However, mission requirements for DSN resources are not trivial to model (Clement and Johnston, 2005).

There has been much work aimed at automating the DSN scheduling process. For many years, the Operation

Mission Planner (OMP-26) used heuristic search to allocate 26-meter antennas to missions and linear programming to adjust track time intervals (Kan et al. 1996). Other automated scheduling tools were research projects and were never deployed. LR-26 is a customizable heuristic scheduling system also for the 26-meter antennas using Lagrangian relaxation and constraint satisfaction search techniques (Bell 1997). The Demand Access Network Scheduler (DANS) expanded the scope to include all antennas using a heuristic iterative repair approach (Chien et al. 1997). These systems schedule mission requirements of the form, "four 15-minute tracks every day." In this paper, we consider an approach that combines some of the strengths of these systems. Other GUI planning tools that have been used for forecasting, analysis, and manual scheduling include TIGRAS (Borden et al. 1997) and FASTER (Werntz et al. 1993).

Other systems have investigated oversubscribed scheduling problems that capture the basic constraints of DSN's mid- to long-term resource allocation. The Air Force Satellite Control Network (AFSCN) also schedules satellite communications requests, on a larger number of satellites and ground stations, but limited to one day at a time. Requirements are more simply specified as an ordered list of resource and time window pairs (Barbulescu, Watson et al. 2004; Barbulescu, Whitley et al. 2004). For this problem, which exhibits "plateaus" in the search to minimize the number of conflicting activities, local repair techniques have been found less effective than approaches which make more moves at once. Another satellite scheduling problem is that of fleets of Earth observing satellites where the activities to schedule have similar kinds of viewperiod constraints, but additionally require onboard resources such as instruments and data recorders (Frank, Jonsson et al. 2001). In this problem, however, the different requests are prioritized, and the goal of finding a "best" subset to fit on the schedule can be addressed with a greedy approach using texture-based heuristics, as in e.g. (Beck, Davenport et al. 1997).

Conclusion

The development of SSS is a unique opportunity to deploy mixed-initiative scheduling system. There is strong support from users to implement this new scheduling application, and automated scheduling is recognized as an essential ingredient to its success. Exploring approaches to meet the requirements placed on SSS has uncovered many general mixed-initiative planning design issues that strongly impact the user's flexibility and control of scheduling. In particular, the interface for managing requests and suggesting changes from a scheduling engine must be carefully designed to avoid taxing the user.

Acknowledgments

The ideas and design decisions for the SSS are a product of many individuals thinking about this problem over many years, of which the authors are but two. Contributions to the SSS have been made by <to be supplied at publication>. 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.

References

Ai-Chang, M., J. Bresina, L. Charest, A. Chase, J. C.-J. Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. G. Chafin, W. C. Dias and P. F. Maldague (2004). "MAPGEN: mixed-initiative planning and scheduling for the Mars Exploration Rover mission." IEEE Intelligent Systems 19(1): 8-12.

Barbulescu, L., J.-P. Watson, L. D. Whitley and A. E. Howe (2004). "Scheduling Space-Ground Communications for the Air Force Satellite Control Network." *Journal of Scheduling* **7**(1): 7-34.

Barbulescu, L., L. D. Whitley and A. E. Howe (2004). "Leap Before You Look: An Effective Strategy in an Oversubscribed Scheduling Problem". AAAI 2004.

Beck, J. C., A. J. Davenport, E. M. Sitarski and M. S. Fox (1997). "Texture-Based Heuristics for Scheduling Revisited". AAAI 1997.

Bell, C. (1992). "Scheduling Deep Space Network Data Transmissions: A Lagrangian Relaxation Approach," Technical Report, Jet Propulsion Laboratory.

Borden, C., Y. Wang, and G. Fox (1997). "Planning and Scheduling User Services for NASA's Deep Space Network," Working Notes of the 1997 International Workshop on Planning and Scheduling for Space Exploration and Science.

Bresina, J., A. Jónsson, P. Morris, and K. Rajan (2005). *Mixed-Initiative Activity Planning for Mars Rovers*, *IJCAI-05*, page 1709.

Chien, S., R. Lam, Q. Vu (1997). "Resource Scheduling for a Network of Communications Antennas," IEEE Aerospace Conference. Aspen, CO.

Chien, S., R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, A. Davies, R. Lee, D. Mandl, S. Frye, B. Trout, J. Hengemihle, J. D'Agostino, S. Shulman, S. Ungar, T. Brakke, D. Boyer, J. VanGaasbeck, R. Greeley, T. Doggett, V. Baker, J. Dohm, F. Ip (2004). "The EO-1 Autonomous Science Agent," Proceedings of AAMAS.

Clement, B.J. and M.D. Johnston. *The Deep Space Network Scheduling Problem*. in *IAAI*. 2005. Pittsburgh, PA: AAAI Press.

Dechter, R., I. Meiri and J. Pearl (1991). "Temporal Constraint Networks." *Artificial Intelligence* 49: 61-96.

Eggemeyer, C., S. Grenander, S. Peters and A. Amador (1997). Long Term Evolution of a Planning and Scheduling Capability for Real Planetary Applications. Working Notes of the First International Workshop on Planning and Scheduling for Space, Oxnard, CA.

Frank, J., et al. *Planning and Scheduling for Fleets of Earth Observing Satellites*. in *I-SAIRAS 2001*. 2001.

Kan, E.J., J. Rosas, and Q. Vu, *Operations Mission Planner - 26M User Guide Modified 1.0*. 1996, JPL Technical Document D-10092.

Muscettola, N., P. Nayak, B. Pell, and B. Williams (1998). "Remote Agent: To Boldly Go Where No AI System Has Gone Before," *Artificial Intelligence* 103(1-2):5-48, August.

Myers, K. L., P. A. Jarvis, W. M. Tyson, and M. J. Wolverton. *A Mixed-initiative Framework for Robust Plan Sketching*, In *Proceedings of the 13thInternational Conferences on AI Planning and Scheduling* , Trento, Italy, June, 2003.

Wilkins, D. E. *Using the SIPE-2 Planning System: A Manual for Version 4.3*, Artificial Intelligence Center, SRI, 1993. International, Menlo Park, CA.

Werntz, D., S. Loyola, and S. Zendejas. *FASTER - A tool for DSN forecasting and scheduling*. in *Proceedings of 9th AIAA Computing in Aerospace Conference*