TARDIS: An Automation Framework for JPL Mission Design and Navigation

Ian M. Roundhill[1] and Richard M. Kelly[2]
Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Dr., Pasadena, CA 91109

Mission Design and Navigation at the Jet Propulsion Laboratory has implemented an automation framework tool to assist in orbit determination and maneuver design analysis. This paper describes the lessons learned from previous automation tools and how they have been implemented in this tool. In addition this tool has revealed challenges in software implementation, testing, and user education. This paper describes some of these challenges and invites others to share their experiences.

I. Introduction

Mission Design and Navigation at the Jet Propulsion Laboratory (JPL) has implemented an automation framework tool named TARDIS (Traceable Automation with Remote Display and Interruptible Scheduler). This ground software tool is intended to facilitate increased automation for orbit determination, maneuver design, and other orbit analysis. While many JPL scenarios require human ingenuity, there are other scenarios that are amenable to automation. One of the first users of this tool will be the Soil Moisture Active Passive (SMAP) orbit determination team. This software is designed to allow users to specify the tasks to be executed and specify the types of events that will trigger the creation of the tasks. The implementation of TARDIS also included testing and implementation challenges that may be common to other software automation efforts. Some of the challenges include duplication of test anomalies, methods for file detection, and integrating automation and security. This paper will provide an overview of this software including the background of previous automation implementations, requirements that we learned from those implementations, implementation details for TARDIS, and testing results.

II. Background

There have been various implementations of automation for orbit determination within Mission Design and Navigation at JPL. Over the last twenty years, there have been six notable instances that were used as examples and lessons for the implementation of TARDIS. These four missions were TOPography Experiment (TOPEX), Gravity Recovery and Climate Experiment (GRACE), Mars Global Surveyor (MGS), and Mars Odyssey, Mars Reconnaissance Orbiter (MRO), and Mars Science Laboratory (MSL).

The TOPEX spacecraft launched in 1992 into low orbit to measure the surface topography of the Earth's oceans[1]. JPL was responsible for orbit determination in support of scheduling, sequencing, and long-term orbit maintenance. There were no specific navigation targets in terms of a specific place at a specific time. Due to the operations scenario and the knowledge of the Earth's environment, the team automated a significant portion of the orbit determination process. They encoded the process into a monolithic script that was invoked via the UNIX cron facility. This system worked well for a two or three member team with no intention of re-using the software for a future mission.

The two GRACE spacecraft have been in a polar orbit measuring the Earth's gravity field since 2002. JPL conducted orbit determination on a daily basis to evaluate the performance of the end-to-end measurement system. A customized application that implements the orbit determination process in multiple tasks was created to support an extremely small number of operators. The application is constantly running and includes internal pauses. After an internal pause, it evaluates which files are ready for processing and which tasks should act on them. This application

[1] Navigation System Engineer, Jet Propulsion Laboratory, California Institute of Technology, MS 301-121, 4800 Oak Grove Drive, Pasadena, CA, 91109.
[2] Engineering Applications Software Engineer IV, Jet Propulsion Laboratory, California Institute of Technology, MS 301-121, 4800 Oak Grove Drive, Pasadena, CA, 91109.

creates the orbit determination solution, evaluates parameters against tolerance values, notifies the operator of any anomalies, and prompts the operator to review the results & finalize the task via a Web interface.

The MGS and Odyssey spacecraft launched in 1996 and 2001 on missions to explore Mars. These teams adopted a similar strategy to coordinate task creation via the Unix make function. This function allowed them to define tasks based on the appearance and creation of files. In addition, users can define branching functionality via the creation of different files based on the outcome of a task. For example, a 'success,txt' file can prompt a subsequent process while an 'error.txt' file can prompt a messaging task.

The MRO spacecraft launched in 2005 and currently orbits Mars collecting global imaging, spectral, and radar data. As the Navigation team learned more about the behavior of the spacecraft and its interactions with Mars, they created a process to execute every morning. This quicklook process was designed to provide analysts with information as they started their daily work. The implementation is a monolithic script that is operated via cron. The recent MSL Navigation team created a very similar process to aid them in their work.

Some common features are evident in these previous implementations of automated orbit determination: the use of cron, the use of make and highly customized implementations. Many of them used the UNIX cron facility via an individual analyst's computer account. Cron is well documented and is easy for an individual user to implement. However, there are a variety of limitations that we were interested in addressing. First, only one team member can administer the cron job. If a user leaves the team or takes an extended vacation, other team members are limited in their ability to administer the automation. Second, files appear to be created by this user, even though they were intended to be created on behalf of the team. If one wants to examine the products created by the team it will be difficult to discern which items were created by the team member via manual processing and which were created by the automation. Third, a convoluted polling scheme is required if the team wants to run the analysis when files appear. One would need to configure the cron facility to invoke software that would evaluate the filesystem. This evaluation would need to reference a database of previously 'discovered' files and be able to append newly discovered files to the database. Cron is not designed to detect the appearance of a file; it is designed to detect the crossing a time boundary. Fourth, if a process takes an extended time to complete, the cron facility may create a second process that might inadvertently interact with the first process.

Other teams used the UNIX make facility. Make is also well documented and many analysts can learn the syntax with some practice and attention. Unfortunately make had a significant drawback because it cannot start itself based on a file appearance. A user needs to invoke make in some fashion for it to evaluate the status of the files. Some users did combine make and cron to create a daily orbit determination solution.

These implementations are also highly customized. They are written by the analysts for their exclusive use and only they know how to operate them. The software expertise behind these implementations varied widely and this fact resulted in uneven code sophistication and documentation. The result was that only operators who were deeply familiar with them could make changes. These implementations were also very difficult to port to a new mission or new task. There were a large number of mission-specific choices and assumptions that were embedded into the implementations. Unfortunately these choices made significant reuse practically impossible and subsequent missions would spend resources (time, money, focus) on a new implementation.

III. Design

As the Soil Moisture Active Passive mission was being designed at JPL, Mission Design and Navigation recognized there was an opportunity to design a multi-mission process automation tool that benefited from the previous experiences. The SMAP mission is an Earth orbiting mission intended to increase our knowledge of the Earth's water. This type of mission is amenable to automation of the orbit determination process because of the high knowledge of the Earth's environment and high observability of orbit due to the tracking station-spacecraft geometries. Each tracking pass observes a significant amount of the spacecraft's argument of latitude as compared to a deep space mission. A parallel Misison Design and Navigation institutional effort to advance automation was also completed as the SMAP design was beginning. A prototype of various automation functions had been created and was demonstrated to navigators. The TARDIS system requirements were derived from the Mission Design and Navigation automation prototype and the SMAP mission design.

Clearly one of the key elements of automation is the ability to create user-defined tasks. The operators need to be able to define what actions need to be taken. For TARDIS, this can be a script stored inside TARDIS or it can be a script or progam stored outside it. In the first case, TARDIS provides a graphical user interface to allow the user to place a script into TARDIS's internal database. When the script is needed, TARDIS places it on a filesystem in a unique directory and executes it. In the second case, the user gives TARDIS a command line string that it will later use to invoke a script or program already present on the filesystem.

We identified three types of events that should be detected by TARDIS and used to create a task. The first is the occurrence of a time. For example, a task can be executed every morning at 5am to execute a routine script. When the operators arrive at work, their work products are ready for evaluation, approval, or analysis. The user interface allows users to input month, weekday, day, hour, and minute fields. Previous automation implementations used cron and many of the operators are familiar with this style.

The second event is the appearance of a file. Files are often delivered onto a filesystem by a previous process executed by another team. One example is when a ground station has completed a tracking pass and has delivered a file. Similarly, a spacecraft team may have finished review of maneuver parameters and placed a file on a fileystem. TARDIS will detect the appearance of the file and create a task to process the file in a pre-determined manner. An example of this event definition is shown in Figure 1. This task will be created whenever the file 'sigsolution.boa' is created in a numbered directory. In addition, the name of the file is passed to the script via the $1 variable specified in the command line entry.

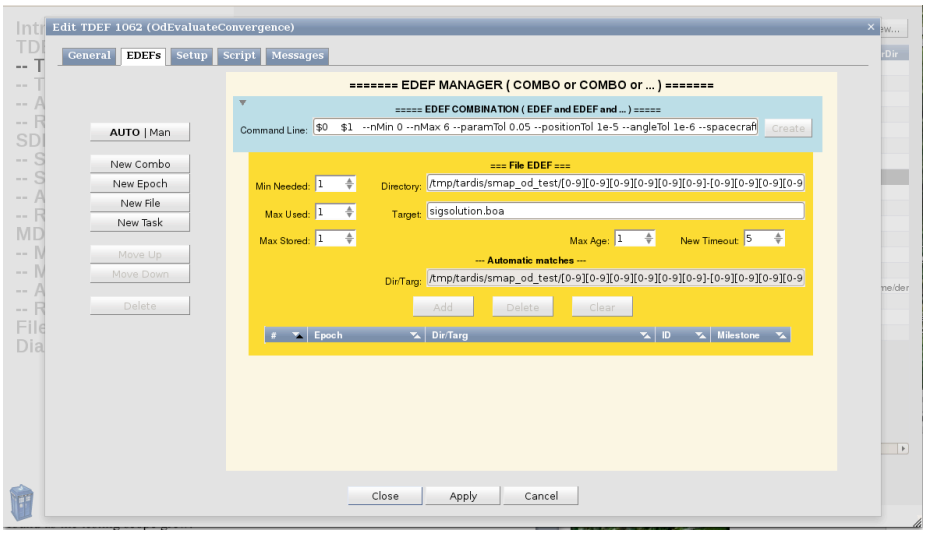

Figure 1: Example of File Event Definition

The third event is the completion of a TARDIS task. Additional options exist to narrow the specification based on the name and the success or failure of the completed task. For example, the operator can request that a task be invoked whenever tasks with the name string 'orbit' complete with an error code. This functionality allows the operators to create sophisticated, custom notification schemes to support their specific mission.

Finally, the operators can mix and match these events. For example, an operator can specify a task to be created when the input file has been delivered and the time is 6 am. This feature allows an operator to implement a process in a set of tasks. These tasks will then be invoked in the correct order based on the results of their completion. TARDIS implements logic to allow complicated nesting and branching of tasks.

TARDIS also allows the user to control specific parameters concerning task execution. The operator can specify the total number of tasks allowed to operate under TARDIS, their UNIX niceness level, and maximum execution time. These parameters can be assigned per-task or globally. For example, the operator can specify that only one task A is allowed to run at any time while three instances of task B are allowed. With these parameters, an operator can ensure that high priority tasks are given preferential treatment and the computer is not accidentally filled with a large number of tasks due to operator or data error. An example of the control parameters at the TARDIS level is shown in Figure 2.

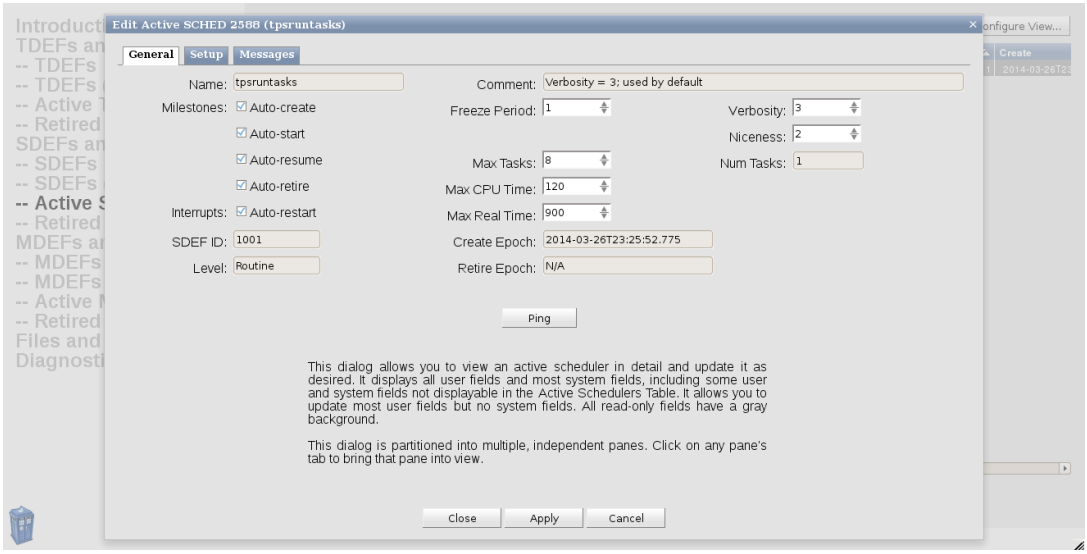

Figure 2: TARDIS Control Parameters

In addition we wanted TARDIS to assist the user in launching tasks in a manual manner. There may be a need for special processing due to inconsistencies in the input data. Even the best laid plans can have unexpected events and the ability to manually create tasks is used for these scenarios. The operator is allowed to enter a task definition and request creation by TARDIS. This feature has been found to be useful during testing. An operator may have a set of scripts to be implemented via TARDIS and there are some integration details to be finalized. The operator will then invoke a task manually, evaluate the output, make changes, and invoke the task manually again. All of these invocations will be stored in the TARDIS database with a record of the inputs and outputs.

Another key element of automation is the need to store information about the task results. When a failure occurs in software operated by humans, diagnosis is typically conducted by visual inspection of output data. An automated process would not have this benefit; a human is not present at the computer. TARDIS needs to record the results and allow for inspection of the artifacts at a later time. This functionality is shown in Figure 3. The operator can review which tasks have been retired, i. e. completed, the corresponding status code, the times of creation and retirement and more.

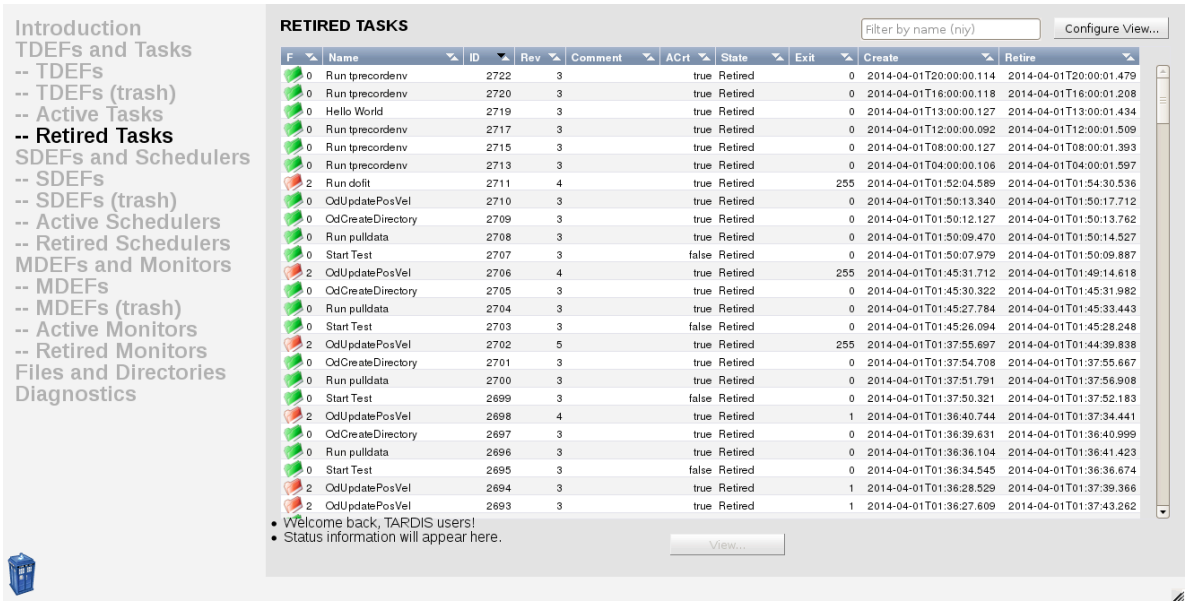

Figure 3: Display of Retired Tasks

TARDIS runs on a host computer that may encounter sudden problems including power failure, hardware failure, or others. If one of these problems occurred, clearly all automation would cease as the computer system administrators address the problem. After the host computer has been returned to service, TARDIS needs to autonomously start itself, address the status of the tasks that were running just before the interruption, and be ready to create any new tasks. This functionality was addressed by running TARDIS as two UNIX daemons whose lifetimes are controlled by UNIX init scripts. Whenever the host computer is returned to service, it will automatically start the TARDIS daemons.

IV. Software Implementation and Challenges

We used object-oriented design throughout and implemented most of TARDIS in C++, with the remainder implemented in Python. The C++ portion comprises the core modules and programs, including (a) the database management system (DBMS) interface, (b) graphical user interface and web capabilities, (c) task definition and execution, (d) event detection and evaluation, (e) monitor commands, (f) scheduler commands, and (g) system administration commands. The Python portion provides a library of utility functions to assist any tasks written in Python.

One important design decision was whether TARDIS's file-based event detection should be polled, event-driven, or some combination of the two. A polling algorithm is easier to implement but has two main disadvantages. It can result in long delays before file-based events are recognized, and it can overload a computer that has large filesystems. An event-driven algorithm is harder to implement but removes these disadvantages. TARDIS implements an event-driven algorithm that detects files written by the system that is hosting TARDIS. The additional challenge of detecting files that are written by other computer systems to a common filesystem is a near-term future improvement. An individual computer knows about the files that it is writing to a common filesystem. It doesn't automatically know about files being written by other computers to a common filesystem. The solution is for the computers to exchange information about the files they are writing.

Another challenge we encountered was the interaction between the goals of increased automation and increased individual computer accountability. Increased automation improves the reliability of simple systems and enables individuals to focus their energies on other challenges. Increased individual computer accountability is used to ensure that computer resources and the records stored within them are used for approved purposes. One result of individual accountability is the restriction in the use of group accounts. We cannot create a computer account and provide its password to multiple people. Nor can we implement a software authentication mechanism that essentially duplicates group accounts. The use of automation on behalf of a team requires attention to the details of both of these goals. We implemented a solution that allowed all of the team members to control the automation. All of the actions of TARDIS are logged and all of the files written by the automation are identified as such. This solution allows the flexibility for the members to act as a team and the accountability to understand the source of each of the files on the filesystem.

V. Deployment of TARDIS

The TARDIS software and associated libraries are deployed on a virtual machine. The virtual machine emulates the behavior of a stand-alone computer and is hosted on a computer with virtual machine management software. This implementation has some significant advantages for Mission Design and Navigation. First, if the routine tasks have small computational demands, we can achieve cost benefits by implementing many virtual machines on one server. Perhaps various teams can stagger their 'daily' solutions at 15-minute intervals across the 6-o-clock hour. There is no need to purchase multiple computers for a small amount of computations. Second, if one virtual machine needs additional computing resources, we can rebalance the virtual machines across multiple servers to ensure that every mission receives appropriate resources.

A low-demand virtual machine and a high-demand virtual machine can be housed on the same server. The use of a virtual machine is not obligatory and we have deployed TARDIS to an individual workstation. There is also an option to deploy multiple TARDIS instances to one computer. Such a deployment requires additional attention to security because all of the operators with access to that computer will have access to the all of the TARDIS instances. This may be encouraged for some missions with various sub-teams and may be disallowed when operators are not allowed to view information from another mission. For example, many Mars orbiter navigators are all on the same teams and are permitted to view each other's work. A counter example is a team that has a foreign national analyst who is only permitted to view a very specific range of information; therefore this team's TARDIS instance needs to be on a separate virtual machine.

VI. Testing and Results

The TARDIS team conducted testing in four campaigns. First, a standard, unit-testing approach was used to test functions, classes, modules, programs, and program sets. This allowed most bugs to be caught as early as possible while eventually ensuring correct operation of multiple programs working together. Second, a set of task definitions were written to test various use cases in simulated user environments, including significant features, edge cases, and stress tests. In addition to helping us catch more bugs, some of these task definitions also served as examples for the User's Guides. Third, the MRO quicklook process was implemented via TARDIS to shadow the operations process. An orbit determination solution was conducted every day and compared to the operations orbit determination process. This test used files from the operations process and allowed users to experience how the automation may interact with their process. Finally, simulated tracking data was created for the SMAP mission. This data was processed through the SMAP TARDIS process. A result of this testing was the detection of various straightforward bugs that were addressed. Other results of the testing are discussed elsewhere in this paper; for example the topic of human evaluation is addressed below and the topic of access control was addressed previously.

The TARDIS system presents challenges for the investigation of testing anomalies. Mission Design and Navigation typically writes single-threaded non-interactive software to implement orbit analysis mathematics. This single-threaded type of software using file-based inputs is deterministic and easily repeatable. When a user encounters unexpected behavior, he can typically copy all of the relevant files to an area where the software developer can immediately duplicate the behavior. TARDIS is quite different. There are a variety of interleaved inputs from the multiple users, operating system, and iterative behaviors within the software and tasks. A one-for-one duplication of software errors can be extremely difficult. The logging system was designed to increase the potential for recording the possible sources of errors and provide information to help the developer find and fix bugs. We found ourselves constantly balancing the need for more logging information to support testing and the need to keep the logging concise to support readability and to avoid overwhelming system resources.

The implementation of an automated process at the start of a mission created a discussion point for the operators concerning the appropriate level of the human evaluation. Previous orbit determination processes used human operators to create and review the solutions. A person was trained to review convergence criteria, check for bad data points, etc. Whenever a task is automated, one needs to spend a non-trivial amount of time thinking, discussing, and deciding on the appropriate level of human intervention. This topic was out of the scope for the software implementation because users can design their processes to have human intervention whenever they want. Any automation tool will benefit from a thoughtful discussion about the appropriate level of human evaluation and control.

VII. Future Uses and Enhancements

The SMAP mission is on track to use an automated orbit determination process for the science mission. The team has evaluated previous automation implementations, created the TARDIS implementation, tested TARDIS in multiple test scenarios, and integrated TARDIS into the mission plan. SMAP will launch in November 2014 and orbit determination will be conducted for a brief few months with significant human oversight. The goal in this period is to tune the parameters and process for the actual SMAP specific environment. During this mission phase, the operators will create manual orbit determination solutions and TARDIS will create automated solutions. These manual and automated solutions will be compared to evaluate the performance of the automation. The spacecraft will transfer into the science orbit and the automated process being run by TARDIS will be used for routine orbit determination.

Other teams within Mission Design and Navigation are evaluating TARDIS. These teams include the Cassini Navigation team, MRO Navigation team, and the Mars and Lunar Collision Avoidance evaluation team[2]. All of these teams have processes that matured over time and may be amenable to automation. The two navigation teams have each operated their spacecraft for over a decade and they have gained significant knowledge about their respective spacecraft and the space environment within they operate. The Mars and Lunar Collision Avoidance evaluation team has the task of calculating close approaches of spacecraft at Mars and the Moon based on trajectories provided by the respective missions. The calculations are fairly straightforward and need to be conducted on a regular schedule. Clearly this is a task that is amenable to automation and this team is currently evaluating TARDIS.

For each deployment above, TARDIS operates within one virtual machine and it is able to detect the appearance of a file that is written on that virtual machine. Typically a file is written by this virtual machine onto a mounted filesystem. We have architected a design to allow TARDIS to detect files that have been written by other computers onto the same filesystem. Oftentimes a team will receive deliveries of files from outside teams via a mounted

filesystem. This feature will allow more flexibility in the automation and will also provide more robustness in the event of hardware or software failure. Unfortunately, this feature did not fall within the scope and resources for this initial deployment. This limitation means that when an outside team needs to deliver a file to the SMAP Navigation team, this file needs to be imported via the local computer that is hosting TARDIS. This can be accomplished via a push from the outside team or a pull from the local computer; either operation causes the TARDIS computer to write the file. This limitation is acceptable for the SMAP team because of the inherent separation of the SMAP Navigation team from the SMAP project repository, and because the SMAP Navigation team is using the Mission Design and Navigation multi-mission infrastructure.[3]

Another future feature is the idea of using mobile phones to monitor the progress of the automated processes. One can imagine a world where a user can launch a mobile application and view simple information. Some example questions from the users are "have my input files been delivered?", "did the processing stop with an error?", and "are my products available for human review?"

Over time we expect the users of TARDIS and other automation tools will become more experienced and use these tools for more of their processes. Mission Design and Navigation employs many people who will experiment in unexpected ways. They sometimes provide scenarios that would have been difficult to predict. We will reach out to our users and encourage anyone who uses automation tools to provide feedback about their positive and negative experiences with these tools.

VIII. Conclusions

Automation is an ongoing process; in this paper we shared our experiences and we'd like to hear about the experiences of others. Mission Design and Navigation has collected the lessons from previous automation efforts and turned them into a multi-mission tool to be used by multiple missions. The challenges have been across many topics including software implementation, testing, and human evaluation of automation products.

As a result of this hard work, we now have a multi-mission tool that operators can use to automate the creation and logging of their tasks. The operators can have confidence that the computer will conduct analyses for them at specified times, when files appear, or when tasks complete. They can show up at work with a head start on the issues that need human intervention and be relieved of the tedium of waiting for the computer to finish. Finally, they can go on vacation with confidence. They know that their fellow operators are fully trained on the use of the automation and have institutional assistance whenever they have questions.

Acknowledgments

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

References

[1] Cangahuala, L. and Muellerschoen, R., Yuan, D., Christensen, E., Graat, E., and Guinn, J., "TOPEX/Poseidon Precision Orbit Determination With SLR and GPS Anti-Spoofing Data", GPS Trends in Precise Terrestrial, Airborne, and Spaceborne Applications, vol 115, Springer Berlin Heidelberg, 1996. p. 123-127.

[2] Berry, D. and Guinn, J., Tarzi, Z., and Demcak, S., "Automated Spacecraft Conjunction Assessment at Mars and the Moon," Proceedings of SpaceOps 2012.

[3] Gerasimatos, D. and A. Attiyah, "Engineering a Multimission Approach to Navigation Ground Data System Operations,"SSC/DLR/AIAA SpaceOps 2012, Stockholm, Sweden: 2012.