Multi-mission Activity Planning for Mars Lander and Rover Missions

Paul G. Backes, Jeffrey S. Norris, Mark W. Powell, Marsette A. Vona
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California
Paul.G.Backes@jpl.nasa.gov

Abstract— The Web Interface for Telescience (WITS) provides downlink data visualization and uplink activity planning for multiple Mars lander and rover missions. WITS also provides a ground data system (GDS) for terrestrial rover operations. The architecture of the WITS system that enables its multi-mission use is described. WITS has been used as the GDS for the Rocky7, FIDO, and Rocky8 rovers at JPL. It was used for command sequence generation for the Mars Polar Lander mission robotic arm and robotic arm camera and will be used for science activity planning in the 2003 Mars Exploration Rover (MER) mission. It is also planned for use in the 2007 Phoenix Mars lander mission and 2009 Mars Science Laboratory (MSL) rover mission. WITS is currently being integrated with the Mission Data System (MDS) for use in the MSL mission and with the Coupled Layer Architecture for Robotic Autonomy (CLARAty) system for use as the GDS for terrestrial technology development landers and rovers.

TABLE OF CONTENTS

1. INTRODUCTION

A pair of tools, the Web Interface for Telescience (WITS) [1], and the Parallel Telemetry Processor (PTeP) [2], have been developed to provide multi-mission ground operations capability for landers and rovers. WITS is used for both Mars mission operations and terrestrial rover operations. PTeP is used for terrestrial rover operations. Multi-mission in the context of this paper means both Mars flight missions and operations for terrestrial rovers.

WITS provides collaborative downlink data visualization and uplink activity planning for Mars lander and rover missions. Downlink data visualization and uplink activity planning are integrated so that users can plan lander and rover activities within downlink data, e.g., select targets to drive to and use the targets as parameters in drive activities. The tool provides collaborative planning so that different users and groups of users can work simultaneously to quickly generate the daily activities for the spacecraft. PTeP provides automated downlink data processing.

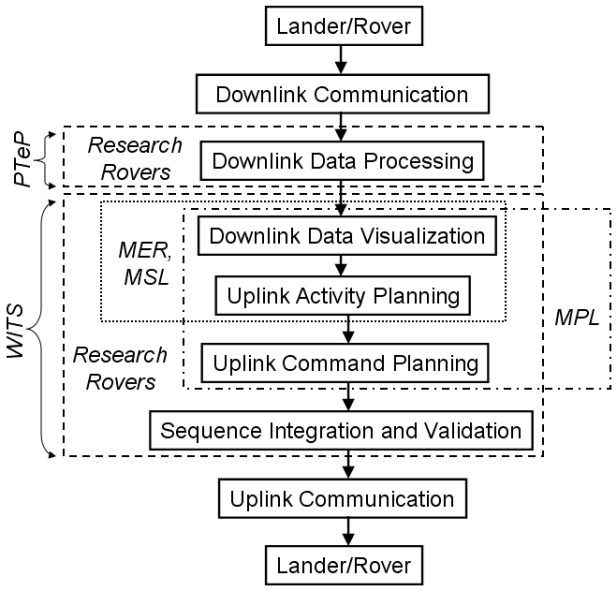


Figure 1. Lander/Rover Ground Operations Process

A simplified ground operations flow diagram for Mars lander and rover missions is shown in Figure 1. The set of tools that are used on the ground (i.e., by operators on Earth) is called the ground data system (GDS). A lander or rover spacecraft downlinks data such as images via Downlink Communication to the ground operations center. Downlink Data Processing processes the data to generate data products that operations tools can use. For example, stereo image pairs are processed to produce range maps, camera models, linearized images, and 3D terrain files. Downlink Data Visualization is used to view the status of the spacecraft and to view the data products. Uplink Activity Planning generates high level activities for the spacecraft. Uplink Command Planning generates the low level commands that will be uplinked to the spacecraft. Sequence Integration and Validation integrates command subsequences to produce the complete command sequence to be uplinked to the spacecraft and validates the sequence. Uplink Communication is subsequently used to send the validated command sequence to the spacecraft.

As indicated in Figure 1, PTeP is used for downlink data processing for terrestrial research rovers. PTeP provides the functionally-equivalent capability of flight mission downlink

0-7803-8155-6/04/$17.00/©2004 IEEE
IEEEAC paper # 1498

Figure 2. Mars Polar Lander at Desert Field Test

Figure 3. MER Mission Rover (170 kg)

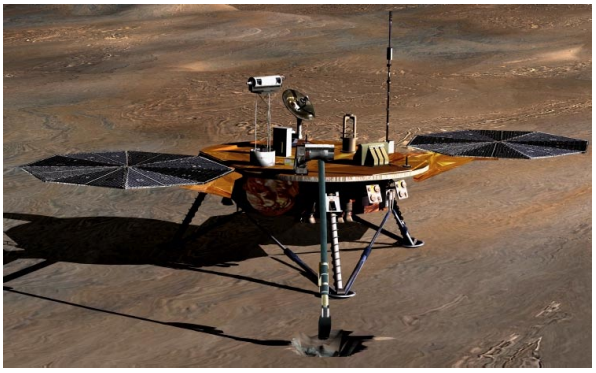


Figure 4. Phoenix Lander

Figure 5. MSL Mission Rover (900 kg)

data processing, but is able to provide this complex functionality at low cost for terrestrial research rover tests.

WITS provides different functionality for different missions and research rovers. WITS was used for Robotic Arm and Robotic Arm Camera command sequence generation for the Mars Polar Lander (MPL) mission which was to begin surface operations in December 1999 [3]. Subsequent to failure to achieve communication with the lander, the operations process was validated in a desert field test, shown in Figure 2. WITS also provided terrain visualization and arm simulation for the MPL mission.

For the 2003 Mars Exploration Rover mission, depicted in Figure 3, WITS is the primary science operations tool for downlink data visualization and uplink science activity planning [1]. The WITS adaptation for MER is called the Science Activity Planner (SAP).

For the 2007 Phoenix Mars lander mission, depicted in Figure 4, WITS will provide terrain visualization and Robotic Arm command sequence generation and simulation.

For the 2009 Mars Science Laboratory mission, depicted in Figure 5, WITS will provide downlink data visualization and science activity planning.

The WITS architecture described in this paper is the third generation of the architecture. The first WITS architecture was used for the Rocky7 (Figure 6) research rover operations and for the 1997 Mars Pathfinder mission public outreach [4]. WITS was run as a Java applet within the Netscape web browser. There were several problems with this architectual approach. First, the user downloaded the applet from the JPL server each time WITS was started, which took too long. Second, the data, such as panorama images, was delivered to the applet over the Internet only when the user opened a view that used the data, which caused more delays. Third, there were difficulties in maintaining compatible versions of Java2D, Java3D, and the web browser.

The second generation WITS architecture was used for FIDO research rover terrestrial field tests [5] (Figure 7) and for the MPL mission [3]. WITS was provided as a Java application that was downloaded onto a users computer once and then run locally to the computer. This eliminated the start-up delay. The database was automatically distributed to each client WITS computer using the Multi-mission Encrypted Communication System (MECS) (formerly called WEDDS) [6].

Figure 6. Rocky7 Rover (15 kg)

Figure 7. FIDO Rover at Desert Field Test (65 kg)

Therefore both the application and data were automatically loaded onto users' computers before they needed them so the application started and accessed data very quickly.

While the second generation WITS solved important architectural problems, there were various implementation approaches that needed improvement. The activity dictionary, which provides definitions of activities available for insertion into a sequence, was specified in a custom textual format. The data views were opened as independent windows which tended to hide previously opened data views and made it difficult for users to manage the data views. The 2D panoramas and 3D terrain visualization views loaded separate image textures which caused excessive memory usage problems. The 3D view used only one level of detail which caused it to use too much memory when providing a high level of detail over a large area.

The third generation architecture of WITS, described in this paper, includes the architectual improvements of the second generation system and provides improved implementations of capabilities. This version is used for the MER mission where it is called the Science Activity Planner (SAP) [1]. The version also supports Mission Data System (MDS) [7], [8] and CLARAty system [9] based rovers. It will also support the Phoenix mission operations system. MDS is the software environment for the MSL mission. MDS is currently used on the Rocky7 rover and will be used on the Rocky8 (shown in Figure 8) rover to test MSL technologies and then on MSL mission rovers. CLARAty is the software environment used for research rovers such as Rocky8 and FIDO. WITS has mission-specific adaptations for each application.

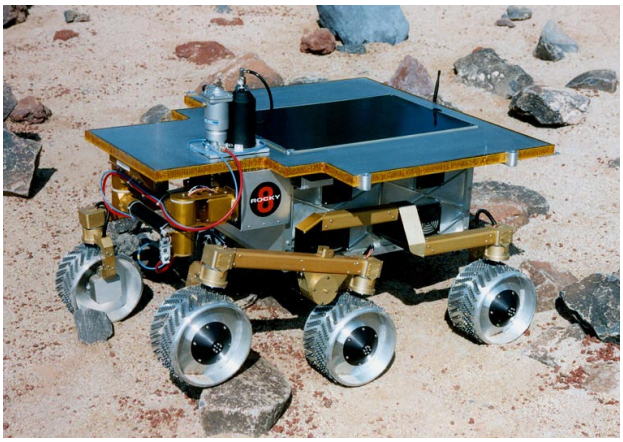


Figure 8. Rocky8 Rover (65 kg)

Several implementation improvements were incorporated in the third generation architecture. Uplink and downlink browsers are used to organize views, as described below, so views do not overlap and get lost. Extensible Markup Language (XML) is used to represent the activity dictionary. Commercial off-the-shelf (COTS) tools are used extensively. There is greatly improved 2D [10], [1] and 3D [11]) visualization compared to the previous implementation.

2. WITS FUNCTIONALITY

Interaction in WITS is done within WITS browsers. The browsers have a selection tree on the left and a work area on the right. There are two primary browsers: the Downlink Browser and the Uplink Browser. The browsers automatically open when a user logs into WITS. The Downlink Browser is shown in Figure 9. The Uplink browser is shown in Figure 10.

Downlink Data Visualization

The Downlink Browser is used to select and view downlink data products. Just like a Web browser can have a list of bookmarks on the left side of the window and the remaining space is for viewing a Web page, the downlink browser arranges links to data products in a tree on the left and creates a view on the right when a link is selected.

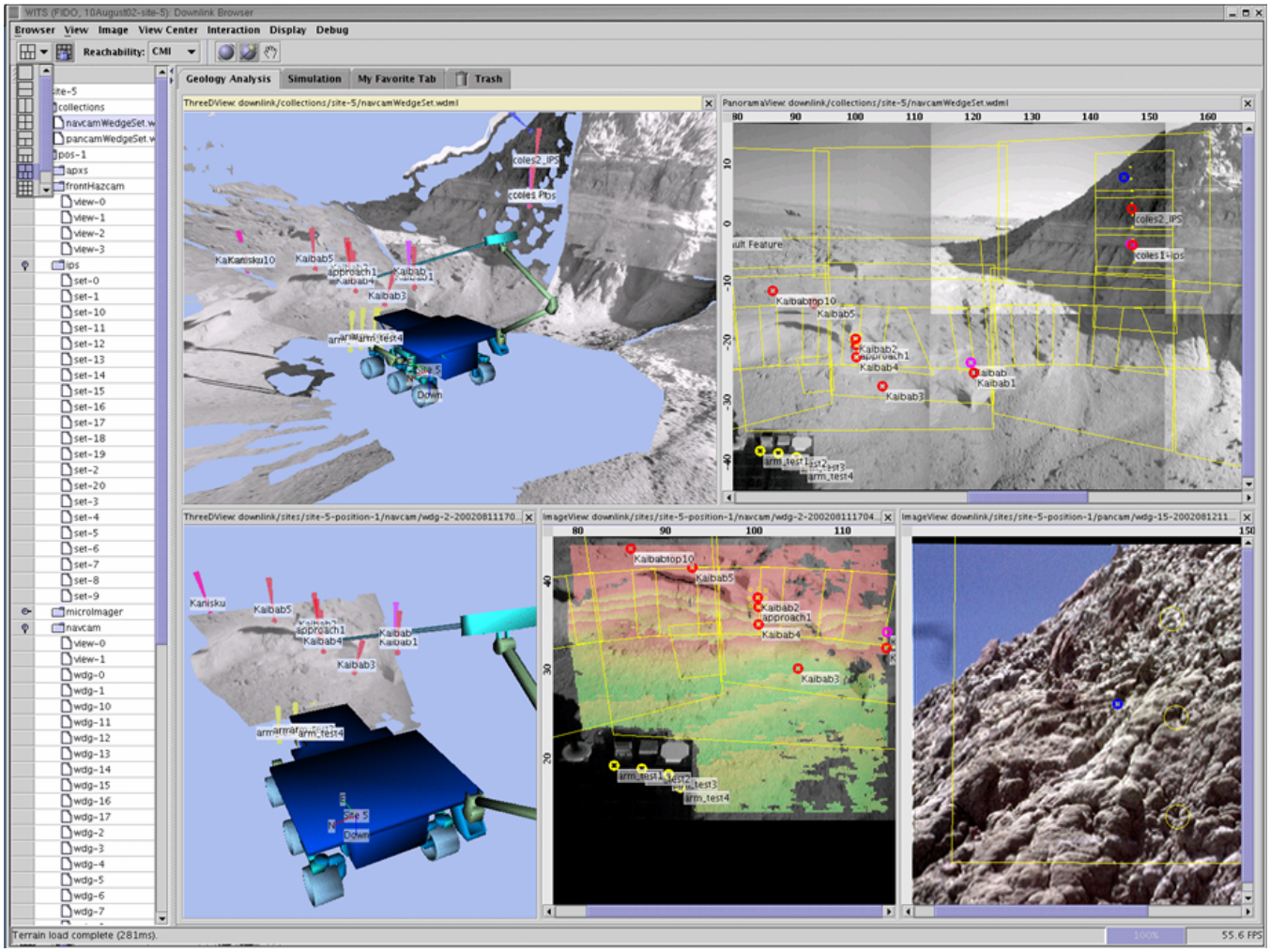


Figure 9. Downlink Browser

The downlink selection tree on the left of the downlink browser contains all of the data products in the rover database. The tree is arranged initially by site, position, and instrument name. The tree may be reorganized by choosing from several combinations of sol, site, position, and instrument ordering.

Selecting data products from the tree to view causes them to appear in the area on the right called the view grid. At first, the view grid takes up all the space in the browser except for the tree. The topology of the grid can be changed by partitioning the viewing area into several parts by selecting a topology option at any time. There are various topologies to choose from, such as 1 by 1, 2 by 2, and 1 over 2.

Additional tabs can be added to the downlink browser to hold additional views. Tabs are useful for organizing groups of views. A new tab is created by right-clicking on a tab area above the view grid and selecting to insert a new tab. Clicking on a tab causes all the views in that tab to be displayed. Tabs can be renamed.

Various types of views, including Panorama and 3D, are provided for viewing data. The Panorama view automatically reads in a collection of images and mosaics them together [10], [1]. Azimuth and elevation angles are displayed. Many features are provided with the Panorama view such as zoom in and out, conversion to anaglyph stereo, and image processing capabilities such as median, low and high pass, gaussian, and edge filters. Glyphs are drawn on the Panorama view representing different planning information. The yellow lines in Figure 9 are footprints representing commanded images in the current plan.

The 3D view displays the 3D terrain and rover, as shown in Figure 9 [11]. Targets, features, and clicked points are also displayed. The 3D view automatically loads terrain segments from a collection of images, e.g., the collection of images in a panorama. Automatic level of detail switching improves display performance. Various navigation capabilities are provided.

To command the rover to drive to or place an instrument on a particular location in the terrain, targets are created. Targets are 3D locations on the terrain that are selected from stereo

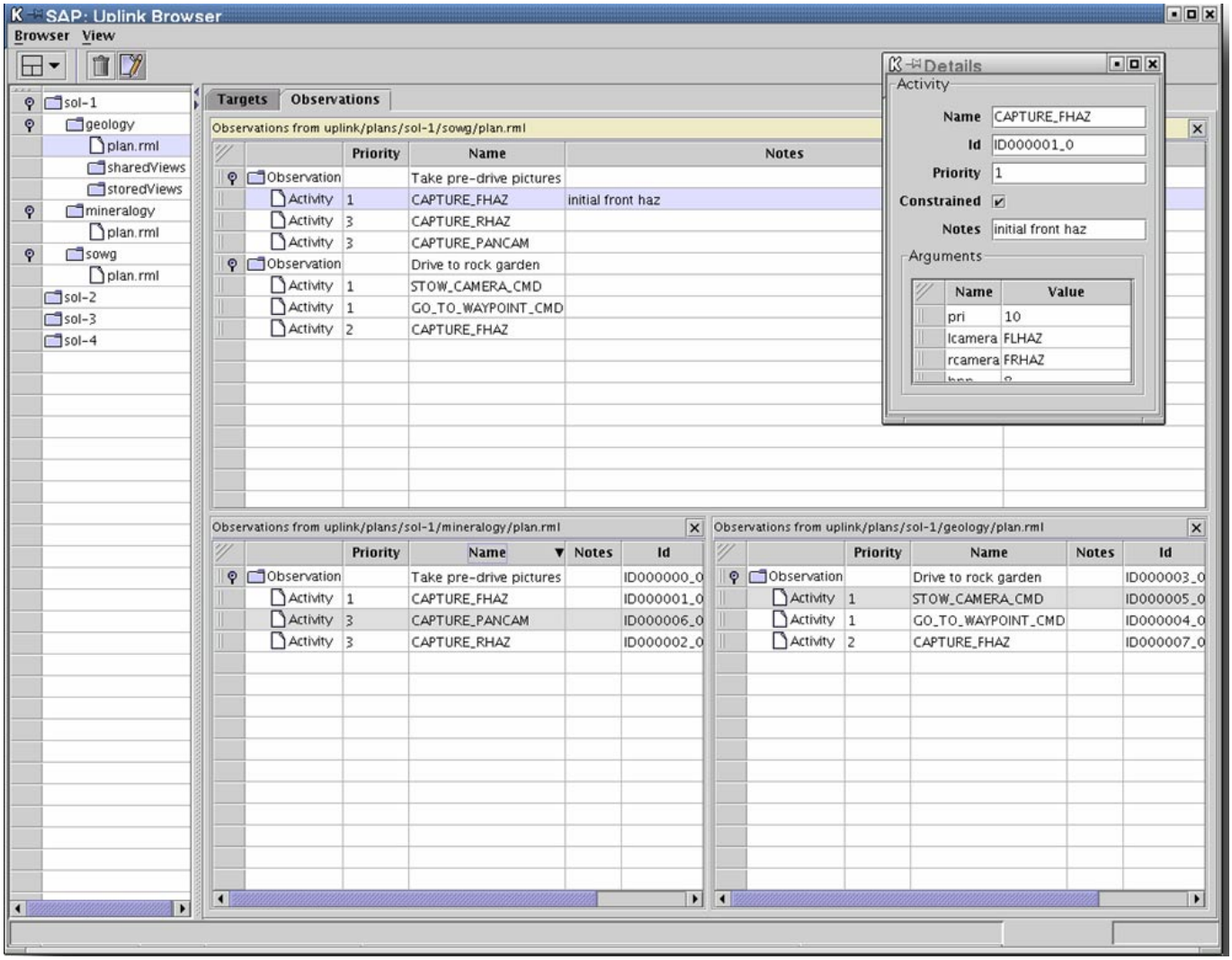


Figure 10. Uplink Browser

image pairs. Features are also 3D locations on the terrain, but they are not used as parameters in activities. Features represent objects in the terrain and targets are associated with features.

After clicking on an image, WITS will try to look up the range of the point in the image. If there is range for the point, then a circular blue annotation, or glyph, is drawn at the clicked point and also in every other open view that contains that point. Glyphs are annotations that are drawn on top of images, like the blue circle, targets, and image footprints.

If a selected point has range data, then a target or feature can be created there via menu items. A dialog prompt will appear for entering a name for the target or feature. For targets, the user also associates the target with a feature by selecting a feature from a pull-down list of features in the dialog prompt. Targets and features appear listed in the Targets view of the Uplink Browser.

Uplink Planning

The Uplink Browser is used to create and edit activity plans, as shown in Figure 10. Activity plans consist of targets, observations, and activities, and are stored in the Rover Markup Language (RML) format, which is based on XML. The left side of the Uplink Browser is an uplink selection tree that allows the user to load a previously saved RML plan. This selection tree is organized by sols and theme groups. A plan is opened from the selection tree by double-clicking on it.

The right side of the Uplink Browser is a WITS view grid. Like the Downlink Browser's view grid, the topology of this area can be selected from a menu of options. The selected topology shown in Figure 10 has one view pane on top and two view panes below. Different plans can be opened in the different view grid view panes.

When the user clicks on an item within a plan (for instance, an activity), details on that item are displayed in a smaller floating window called the Details Dialog (see Figure 10, upper right). The details dialog allows the user to edit attributes of

the currently selected item. The Details Dialog can be shown or hidden using the icon on the Uplink Browser toolbar that looks like a pencil on paper.

Plan merging is accomplished using observation and activity click and drag. Observations and activities can be copied from one plan into another plan by clicking on an element and dragging it into another plan. When an observation is dragged from one plan into another plan, all of its activities are copied with it.

The activities in the plan are visualized in the Panorama and Image views when possible using activity glyphs. For imaging activities, the activity glyphs are footprints on the terrain where the images will be taken. Glyphs for the images in the current plan are shown in Figure 9 with yellow outlines. Glyphs for a vertical spectrometer scan are also shown in Figure 9 as yellow circles with the same angular extent as the spectrometer.

State simulation is provided to assist in evaluating a plan. In state simulation, when the user selects an activity in the plan, the rover position and configuration is updated in the 3D view with the predicted rover state at the end of that activity. A user can quickly click through the plan to get an idea of what the rover is going to do.

3. SOFTWARE ARCHITECTURE

Software Organization

WITS is written entirely in the Java programming language as a Java application. Java applications are implemented as a set of code files called "classes," each of which can be instantiated into an object at runtime. Java classes are typically organized in nested directories called "packages." WITS consists of approximately 650 classes divided into 150 packages. There are "core" packages that are used by multiple missions and mission-specific packages. Capabilities are developed as generically as possible for efficient development of the core packages and to maximize the percentage of the system within the core packages.

The vast majority of the WITS classes are responsible for functions that are not specific to any particular mission's needs. These classes, and the packages they are located in, make up "Core WITS" (Figure 11). The remaining classes are devoted to capabilities that are only applicable to a particular mission, i.e., which must be re-implemented for each mission to account for the differences between them. The term "mission" used here refers to an actual Mars flight mission or a research effort that is fielding a vehicle on Earth. By concentrating as much development as possible in Core WITS, multiple missions benefit from each other's investment in WITS.

The WITS packages (core and mission-dependent) are further divided into three main categories, as listed below.

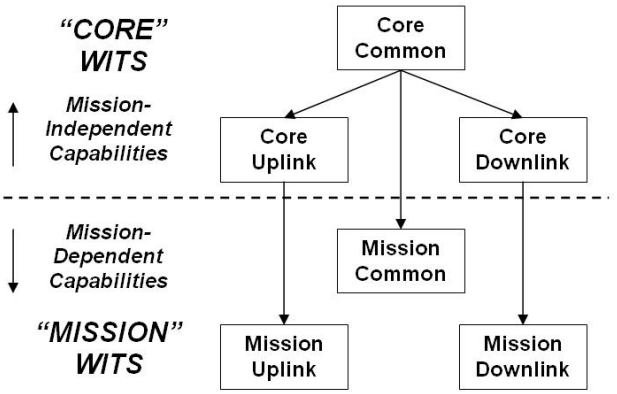


Figure 11. Package Inheritance Diagram

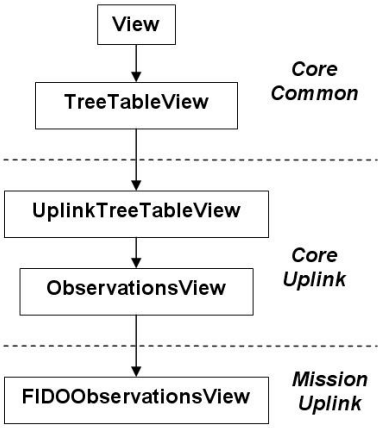


Figure 12. Ancestry of a Mission-specific Class

1. Uplink- Functions specific to the generation of activity plans for transmission to the spacecraft.
2. Downlink- Functions responsible for the visualization and analysis of data acquired by the spacecraft.
3. Common- Functionality that is applicable to both the Uplink and Downlink packages of WITS. Includes base classes for most of the classes in the Uplink and Downlink packages.

Figure 11 illustrates the high-level organization of WITS packages into the categories described above. Classes in the Core Common package serve as parents for classes in the Core Downlink and Core Uplink packages. Each of the Core packages contains classes that serve as parents for classes in the three mission-dependent packages.

An example of the ancestry of one mission-specific class is shown in Figure 12. At the top of this diagram is the View class, which is the base class for over 20 classes in WITS devoted to the display of uplink and downlink data loaded from disk to a user. View is extended by TreeTableView, which provides a graphical display of hierarchical information with spreadsheet-like columns. TreeTableView is extended by both Uplink and Downlink views, but in Figure 12 we focus next on its extension to UplinkTreeTableView, which adds undo/redo capabilities, and ObservationsView, which is fo-

cused on those features particular to the construction of an activity plan for execution by a spacecraft. Some WITS missions simply use ObservationsView as their activity editing interface, but the FIDO research rover requires that its activity plans be written in a particular format that is different from the formats used by the rest of the WITS missions. To provide a menu item to allow a user to request that an activity plan be written out in this format, a simple extension of ObservationsView called FIDOObservationsView was created.

Preventing Cross-Mission Impacts

Sharing of code across multiple missions is a benefit of WITS, but care must be taken to reduce the likelihood that a change made in support of one mission has a negative effect on another. For instance, consider that mission A is a Mars flight project in the final stages of development. Meanwhile, mission B is a research task dedicated to exploring advanced, experimental rover technologies. It would be unacceptable for mission B to make experimental changes to WITS code that could endanger the stability of mission A, but we prefer to not restrict mission B's development for the sake of mission A.

Two rules for WITS development ameliorate the aforementioned problem. First, mission-dependent code for different mission must never interact. A mission-dependent class added for mission B cannot depend, directly or indirectly, on a mission-dependent class for mission A and vice-versa. This way, if mission B adds a feature in its mission-dependent development area, there is no chance that it will have a negative effect on mission A. Second, Core WITS must not directly depend on a mission-dependent class. This rule is a bit more tricky than the last. In Java, a class X typically references another class Y that it depends upon in an explicit, direct manner. This direct reference causes class Y to be automatically compiled when class X is compiled, and furthermore requires that class Y be present whenever class X is used. In our example above, if Core WITS directly depended on one of mission B's classes, then mission A would be forced to include mission B's code, which would expose the flight mission to any bugs that may exist in the research task's class. To rephrase this rule, Core WITS must be able to be compiled in absence of any mission-dependent classes.

The second rule leads to the question "How then, can Core WITS compile for a specific mission without directly referencing mission-specific code?" Two methods are used. First, there is a special Core class called "Mission" that each WITS mission must extend. This class is a gateway to numerous mission-dependent functions and serves as a checklist for adapting WITS to a new mission. At a minimum, a new mission must provide a working implementation for every capability of the "Mission" class. At runtime, WITS asks the Core Mission class to return the specialized version of the Mission class for the currently active mission. This is accomplished without a direct reference to the specialized Mission class through the use of a feature of the Java programming language called introspection. Note that introspection is only needed to bridge the gap to the specialized version of the Mission class since that class is free to directly reference any mission-dependent classes since it is itself a mission-dependent class.

The second method used to access mission-dependent code from Core WITS without a direct reference capitalizes on WITS use of the Castor XML binding framework. Castor is an open source data binding framework for Java (www.castor.org). Castor automatically maps an XML document on disk to a set of WITS Java classes. Typically, all of these classes are in Core WITS, but Castor's default mapping behavior can be overridden with a mission-dependent mapping file that is loaded by the specialized version of the Mission class mentioned above. By providing a mapping file, a mission can cause an element of an XML document to be mapped to a mission-dependent class instead of a Core WITS class.

4. DOWNLINK DATA PROCESSING

A challenge for providing WITS to multiple missions is to adapt WITS to use the downlink data products from the multiple missions. Downlink data differs in content and form for different spacecraft and also between different instruments of a given spacecraft. A convenient high-level downlink data abstraction is the "dataset", which collects data at a given point in time during spacecraft operations. With instrument data, this generally means the product of a given instrument such as images or spectra and the metadata that describes the state of the instrument and the spacecraft at the time of acquisition. In the case of images, WITS groups a set of images, their metadata, and any derived data products that are useful to construct from the raw images together into an ImageContent. The ImageContent object follows the general WITS design of Content in which it is constructed and marshaled as an XML file that can at any time be unmarshaled into the application and put to use. The ImageContent contains the locations in the database of all image data (generally in the form of files on a filesystem) and all known metadata about the instrument and spacecraft at the time of acquisition, as well as derived data products from the original images. Examples of derived data products are radiometrically calibrated images and 3D terrain reconstructions from stereo image pairs.

Raw spacecraft telemetry itself is compressed as much as possible to facilitate the low bandwidth available in actual remote operations away from the Earth. Creating data products that are human readable is therefore a practical necessity. Also, the time constraints of the daily mission operations cycle for missions to Mars such as Pathfinder and MER benefit from organizing these data products into a systematically searchable database.

The Parallel Telemetry Processor (PTeP) is used to create and maintain the WITS database. There are two distinct steps: creating the files in the database and cataloging the files.

PTeP is used to create the files and a separate part of PTeP, the DataStateManager (DSM), is used to catalog the files for a specific mission. PTeP is used to create the database files for terrestrial rover applications, but other tools are used for this function in flight missions. The DSM is used for both terrestrial applications and flight missions.

As WITS has a generalized design that enables it to operate in a mission-independent fashion on the whole but to employ mission-dependent code where needed, so does PTeP have a core processing component which performs the task of reading raw downlink data, passing it through a sequence of processing stages, and organizing the resulting products into a database. The sequence of processing stages may be arbitrary in number and may produce any number of data products from the original input data. The database that PTeP creates currently takes the form of a hierarchical filesystem where each data product is stored as a separate file.

The database files are grouped into ImageContent datasets by the DataStateManager (DSM). The DSM is responsible for creating ImageContent XML files that define the datasets that are viewable from WITS. It also produces MissionState XML files that encode the spacecraft and instrument metadata that is associated with the images, and TableOfContents XML files that allow WITS to present the datasets on the filesystem as an optimized, queriable database. The TableOfContents in the WITS Downlink Browser can be re-sorted very quickly to present datasets in order of time, location, or instrument, or combinations thereof.

The mission-dependent functionality of PTeP is needed if a given mission has specifically defined formats for its data products that must be parsed in a unique fashion. Missions often inherit data product format conventions. For the FIDO and CLARAty/Rocky8 missions, the telemetry format is very similar.

Some missions involve the use of other software systems that perform part or all of the telemetry processing. For instance, the MDS/Rocky7 mission takes in the raw telemetry and stores the resulting products in a relational database. It does not create all of the desired derived data products for activity planning, such as range maps for targeting and 3D terrain reconstruction for arm motion planning and traverse analysis. For this mission, the PTeP processor queries the database for a set of products over a particular time interval. These results take the form of a binary query results file (BQR). The BQR file is then separated into datasets, and PTeP derived related data products from the original images and stores them all on a filesystem. Finally, it runs the DSM on the files to create the XML objects for use in WITS.

In the case of MER, the mission chose to perform all of its downlink data processing with other tools. Therefore only the DSM is needed to define the XML objects for use in WITS.

Implementation of PTeP in the Java language enables computer platform-independence with certain qualifications. Often, the stages of data processing for a given type of instrument data are defined by calling a sequence of existing tools which may be written in a different language. For example, a stereo correlation program that produces a range map and a terrain generation utility that produces triangulated terrain maps are written in the C language. These utilities are called from the PTeP processing control loop that is implemented in Java. It can be used on any system that supports Java, provided that all of the tools that are needed to create the full range of required data products for operations planning are also available for that platform.

5. CONCLUSIONS

The architecture of the WITS system has been developed to enable WITS to be used for multiple terrestrial rovers and Mars lander and rover missions. For terrestrial rovers WITS, along with PTeP, provides a complete ground data system. For Mars lander and rover missions, WITS provides a combination of downlink data visualization, activity planning, and command generation depending on the needs of the mission. The code has been organized to maximize common functionality across multiple missions.

ACKNOWLEDGEMENTS

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

[1] Paul G. Backes, Jeffrey S. Norris, Mark W. Powell, Marsette A. Vona, Robert Steinke, and Justin Wick. The science activity planner for the mars exploration rover mission: Fido field test results. In *Proceedings IEEE Aerospace Conference*, Big Sky, Montana, March 8-15 2003.

[2] Jeffrey S. Norris, Paul G. Backes, and Eric T. Baumgartner. PTeP: The parallel telemetry processor. In *Proceedings IEEE Aerospace Conference*, Big Sky, Montana, March 2001.

[3] Paul G. Backes, Kam S. Tso, Jeffrey S. Norris, Gregory K. Tharp, Jeffrey T. Slostad, Robert G. Bonitz, and Khaled S. Ali. Internet-based operations for the mars polar lander mission. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2025–2032, San Francisco, California, April 2000.

[4] Paul G. Backes, Kam S. Tso, and Gregory K. Tharp. The web interface for telescience. *Presence*, 8(5):529–537, October 1999.

[5] Paul G. Backes, Jeffrey S. Norris, Mark Powell, Kam S.

Tso, Gregory K. Tharp, and P. Chris Leger. Sequence planning for the fido mars rover prototype. In *Proceedings IEEE Aerospace Conference*, Big Sky, Montana, March 8-15 2003.

[6] Jeffrey S. Norris and Paul G. Backes. Wedds: The WITS encrypted data delivery system. In *Proceedings IEEE Aerospace Conference*, Big Sky, Montana, March 2000.

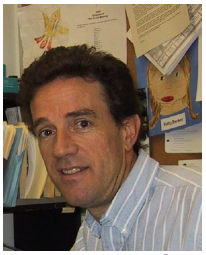
[7] D. Dvorak, R. Rasmussen, G. Reeves, and A. Sacks. Software architecture themes in jpl's mission data system. In *Proceedings IEEE Aerospace Conference*, Big Sky, Montana, March 2000.

[8] D. Dvorak, R. Rasmussen, and T. Starbird. State knowledge representation in the mission data system. In *Proceedings IEEE Aerospace Conference*, Big Sky, Montana, March 2001.

[9] I.A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin. Claraty and challenges of developing interoperable robotic software. In *IROS*, Nevada, October 2003.

[10] Mark W. Powell, Paul G. Backes, Marsette A. Vona, and Jeffrey S. Norris. Visualization of coregistered imagery for remote surface operations. In *Proceedings IEEE Aerospace Conference*, Big Sky, Montana, March 8-15 2003.

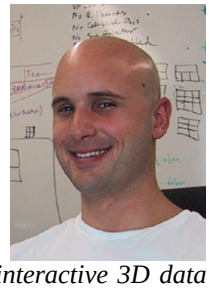
[11] Marsette A. Vona, III, Paul G. Backes, Jeffrey S. Norris, and Mark W. Powell. Challenges in 3d visualization for mars exploration rover mission science planning. In *Proceedings IEEE Aerospace Conference*, Big Sky, Montana, March 8-15 2003.



Paul Backes *is a technical group leader in the Mobility Systems Concept Development section at the Jet Propulsion Laboratory, Pasadena, CA, where he has been since 1987. He received a BSME degree from U.C. Berkeley in 1982, and MSME in 1984 and Ph.D. in 1987 in Mechanical Engineering from Purdue University. Dr. Backes received the 1993 NASA Exceptional Engineering Achievement Medal for his contributions to space telerobotics (one of thirteen throughout NASA), 1993 Space Station Award of Merit, Best Paper Award at the 1994 World Automation Congress, 1995 JPL Technology and Applications Program Exceptional Service Award, 1998 JPL Award for Excellence and 1998 Sole Runner-up NASA Software of the Year Award. He has served as an Associate Editor of the IEEE Robotics and Automation Society Magazine.*

Jeffrey S. Norris *is a computer scientist and member of the technical staff of the Mobility Systems Concept Development section at the Jet Propulsion Laboratory. At JPL, his work is focused in the areas of distributed operations for Mars rovers and landers, secure data distribution, and science data visualization. Currently, he is a software engineer on the Mars Exploration Rover ground data systems and mission operation systems teams. Jeff received his Bachelor's and Master's degrees in Electrical Engineering and Computer Science from MIT. While an undergraduate, he worked at the MIT Media Laboratory on data visualization and media transport protocols. He completed his Master's thesis on face detection and recognition at the MIT Artificial Intelligence Laboratory. He lives with his wife, Kamala, in La Crescenta, California.*

Mark W. Powell *has been a member of the technical staff in the Mobility Systems Concept Development section at the Jet Propulsion Laboratory, Pasadena, CA, since 2001. He received his B.S.C.S. in 1992, M.S.C.S in 1997, and Ph.D. in Computer Science and Engineering in 2000 from the University of South Florida, Tampa. His dissertation work was in the area of advanced illumination modeling, color and range image processing applied to robotics and medical imaging. At JPL his area of focus is science data visualization and science planning for telerobotics. He is currently serving as a software and systems engineer contributing to the development and operation of science planning software for the 2003 Mars Exploration Rover mission. He, his wife Nina, and daughters Gwendolyn and Jacquelyn live in Tujunga, CA.*



Marsette A. Vona, III *is a computer scientist, software engineer, and electromechanical hardware developer. He is a member of the technical staff of the Mobility Systems Concept Development Section at the Jet Propulsion Laboratory. At JPL, his work is currently focused in the areas of high-performance interactive 3D data visualization for planetary exploration, user-interface design for science data analysis software, and Java software architecture for large, resource-intensive applications. Marsette received a B.A. in 1999 from Dartmouth College in Computer Science and Engineering, where he developed new types of hardware and algorithms for self-reconfigurable modular robots. He completed his M.S. in 2001 at MIT's Precision Motion Control lab, where he developed a high-resolution interferometric metrology system for a new type of robotic grinding machine. Marsette*

was awarded the Computing Research Association Outstanding Undergraduate Award in 1999 for his research in Self-Reconfigurable Robotics.