

Mars Science Laboratory Flight Software Boot Robustness Testing Project Report

Brian Roth (Space Grant)
Danny Lam (Mentor, Jet Propulsion Laboratory)

August 15, 2011

On the surface of Mars, the Mars Science Laboratory will boot up its flight computers every morning, having charged the batteries through the night. This boot process is complicated, critical, and affected by numerous hardware states that can be difficult to test. The hardware test beds do not facilitate testing a long duration of back-to-back unmanned automated tests, and although the software simulation has provided the necessary functionality and fidelity for this boot testing, there has not been support for the full flexibility necessary for this task. Therefore to perform this testing a framework has been build around the software simulation that supports running automated tests loading a variety of starting configurations for software and hardware states. This implementation has been tested against the nominal cases to validate the methodology, and support for configuring off-nominal cases is ongoing. The implication of this testing is that the introduction of input configurations that have yet proved difficult to test may reveal boot scenarios worth higher fidelity investigation, and in other cases increase confidence in the robustness of the flight software boot process.

Introduction

The Mars Science Laboratory (MSL) flight software (FSW) boot robustness testing is a task to develop functionality that provides an effective method of comprehensively testing the FSW boot process through a set of possible scenarios. The work on this task was performed by three summer interns with help and guidance from the MSL FSW team, but this report will detail only some aspects of the development. This report will present the life cycle of the project with an emphasis on the architecture of the testing framework and the integration of components from the multiple team members.

Background

Mars Science Laboratory (MSL) is a rover from NASA's Mars Exploration Program, launching in late 2011, that will evaluate if the environment of Mars is today or was in the past able to sustain microbial life. The MSL flight software is used not only to facilitate in-situ experiments on the surface of Mars, but also to support other mission phases including: launch, cruise, and entry-descent-landing. A critical component of the flight software is the boot process, which occurs every time that the computers are turned on. During cruise and descent the computers are expected to be continually awake, thus limiting the number of boots, but during surface operations it is intended that the computers be turned off every night, meaning that the flight software will be booting many times throughout the surface mission. Because of the large number of possible boot scenarios and the potential severity of faults, it is desired that a tool be created to facilitate the testing of flight software booting under a variety of different hardware and software scenarios.

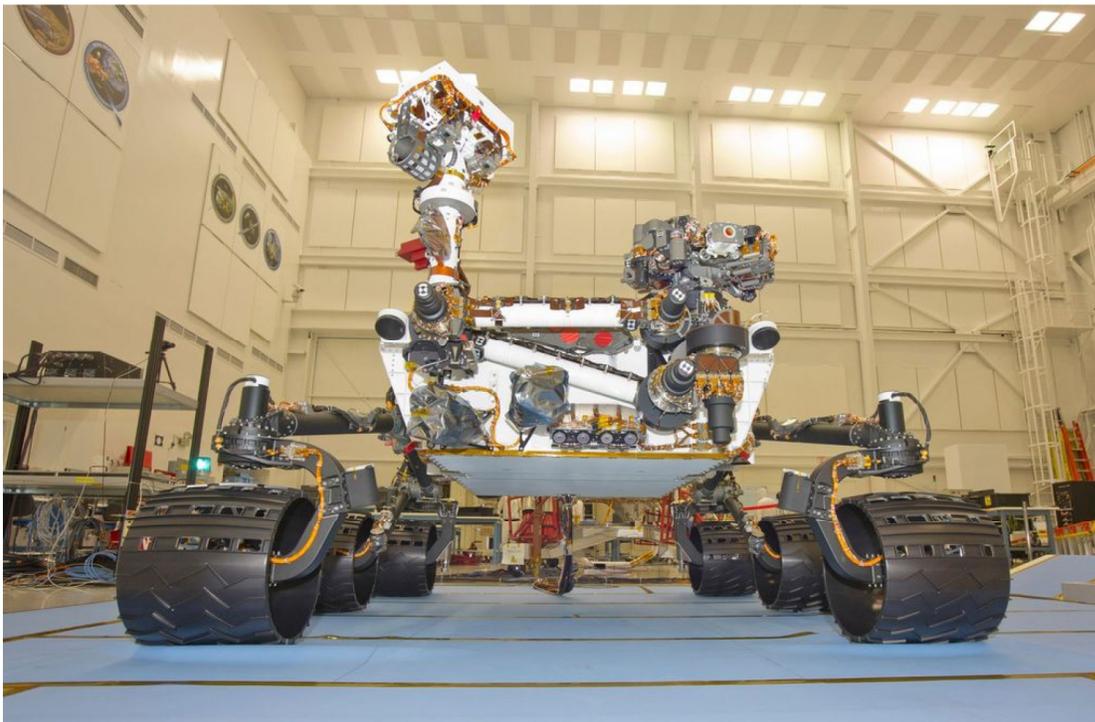


Figure 1: Image of NASA's Mars Science Laboratory taken during mobility testing in the Spacecraft Assembly Facility at the Jet Propulsion Laboratory on June 3, 2011. Image Credit: NASA/JPL-Caltech

Approach

Research:

Throughout the project there has been a need to become familiar with the standard testing procedures and tools used for flight software testing. There has been a need to learn about boot-related hardware, wakeup conditions, and software states, especially at the beginning of the project to understand the objectives of the boot robustness task. During this initial research phase the viability of the software simulation as the base for this testing has been established. The functionality of the Work Station Test Set (WSTS) tool using the Software Simulation Equipment (SSE) to simulate hardware has been determined to have the ability to run the tests and some of the capabilities needed to configure the tests.

Architecture and Design:

As more is learned about how the simulation tools work and how the boot testing tool will be used, the testing framework has been designed and modified. The architecture has not been changed much since the original design, although some details have been worked out as components were implemented, integrated together, and tested. Figure 2 shows a diagram of the architecture for this boot tester, illustrating the interaction among the scripts, files, and memory data.

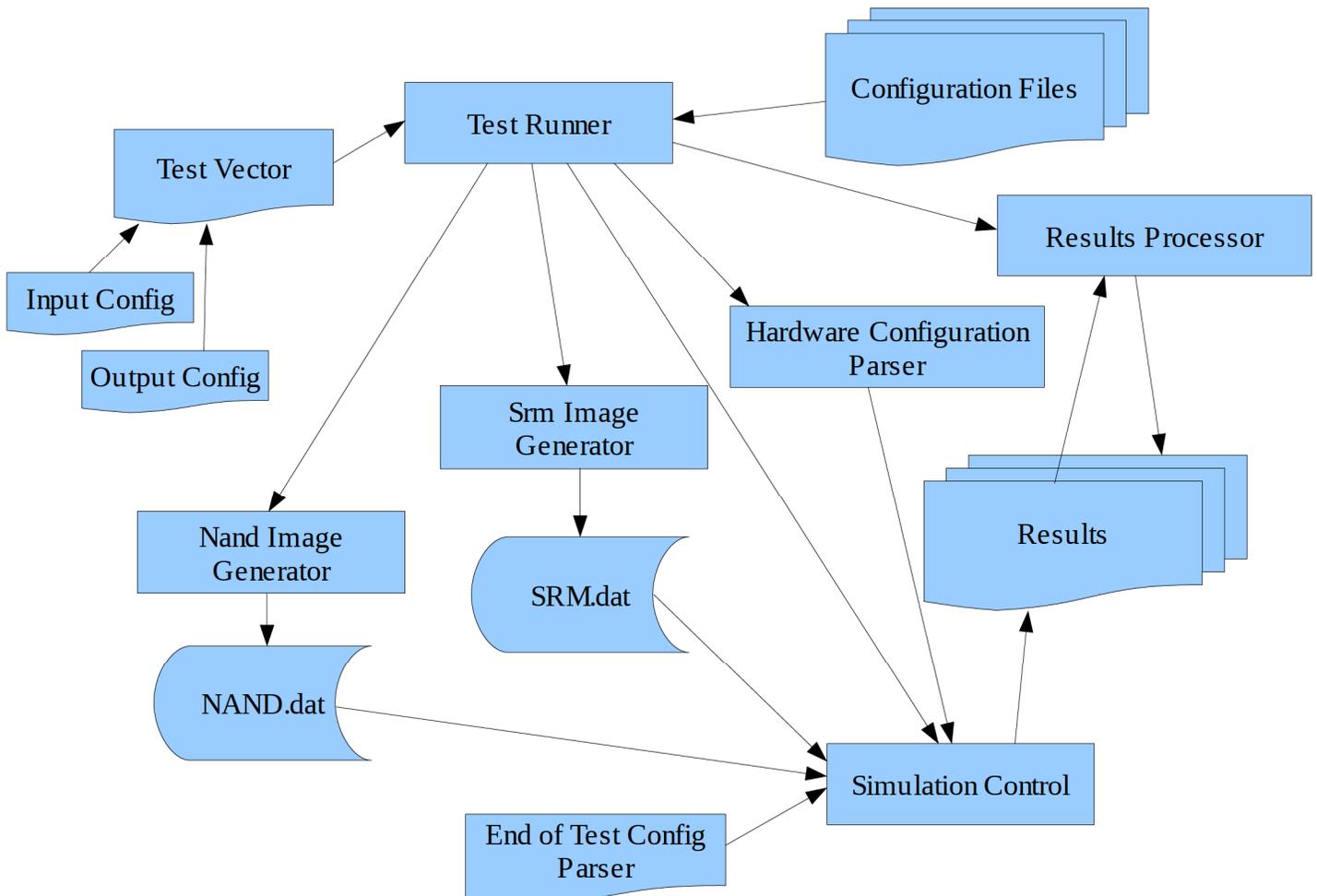


Figure 2: Diagram of the Boot Testing Framework showing the interaction among scripts and data.

There have also been some implementation-related considerations that took some time to figure out. General use of the simulation tools is in an interactive mode with windows open for sending flight software and simulation commands and with flight software running indefinitely until a user-specified end. However for this tester the process has been automated, so the tests are run for a specific number of simulated ticks, which keeps track of simulation time and a socket has been opened to the simulator to send commands after the boot to acquire the end of test hardware state. During this transition there have also been some issues encountered, one of which was caused by specifying the output directory as a location outside of the usual temporary directory. The implication of this is that all file accesses occur across the network, slowing down the simulation, and before finding the correct option to increase the timeout, the simulation was ending prematurely as a result of this. Despite the changes necessary to have the simulation run in the method desired for boot testing, the number of modifications and additions on top of the already available testing tools is surprisingly minimal – with most of the difference resulting from the capability to take specially-designed configuration files to define test scenarios.

Integration:

Aspects of this project have been easily broken down into components, which has allowed parts to be developed almost independently of others. While the actual implementation of a component is not then restricted by the test framework, there has been an understanding of what the interfaces among components will be so that there is consistency and so that when components are integrated together there will not be incompatible interfaces. Adopting a system of scripts with information shared by environment variables and command line arguments in a simple and consistent way has proven successful for this testing and allows the tools to be run piecewise with various functionalities under different environments. By originally trying to keep only one version of up-to-date scripts in a common location, we have learned the utility of having a configuration management tool take care of version as it was often the case that unfinished changes by one person on a script would corrupt results of a test run by a different person. We have since moved the tester into the control of the flight software versioning system, and that has been proving to be a worthwhile improvement.

Iteration:

Along with integration, after our first tests with the tool, the team has been iterating with new functionality, attempting to maintain a working version of the tester while adding new capabilities to have an ever-increasing set of working test configurations. A major capability that has been added, which greatly increases the possible test sets, is the configurability of two image files, which simulate non-volatile flash memory and volatile shared memory. Together these image files provide the capability to specify the software-implied health state of any hardware device and the spacecraft mode and configuration as understood by flight software. These software states have a profound impact on the procedures during the boot process, and testing with off-nominal configurations is desired.

Conclusions

The expectations of a successful project were that the introduction of input configurations that have yet proved difficult to test may reveal boot scenarios worth higher fidelity investigation, and in other cases increase confidence in the robustness of the flight software boot process. The outcome thus far is perhaps just short of that goal, meaning that we have demonstrated the capability to run the tests to increase confidence in boot and perhaps reveal anomalies, but that testing is ongoing. However, the

work of the project is currently being preserved under configuration management and the development is being finished off and cleaned up so that it may become a low-maintenance tool for continued use by the MSL flight software team. Thus this project has resulted in a useful contribution to the MSL project and hopefully with the remaining time a more comprehensive set of tests can be run and the results carefully analyzed.

Acknowledgments

Three summer interns in the Guidance and Control Flight Software Validation group have been developing this functionality with help and guidance from the MSL FSW team and the MSL FSW Internal Test (FIT) team. The details of the task and the desired functionality were specified by the FSW team; simulation tools to incorporate into the testing were provided by the FIT team; and the architectural outline and low-level design suggestions were given by the projects primary facilitator from the MSL FSW team Steve Scandore.

I would like to thank: my mentors John Lai and Danny Lam for the great internship this summer; Steve Scandore for his great leadership and guidance with the task; Dee Leang, Katie Weiss, Ben Cichy, Cindy Oda, Rajeev Joshi, and everyone from the MSL flight software team for their assistance throughout the summer; and the Jet Propulsion Laboratory and NASA Space Grant Program for the great experience this summer.

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by NASA Space Grant and the National Aeronautics and Space Administration.