

# Interoperable Solution for Test Execution in Various I&T Environments

Magdy Bareh<sup>1</sup> and Young H. Lee<sup>2</sup>

*Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109*

**Dawn, the first spacecraft ever planned to orbit two different bodies after leaving Earth, will orbit Vesta and Ceres, two of the largest asteroids in the Solar System. The two asteroids, located between Saturn and Jupiter, have very different properties from each other. By observing both with the same set of instruments, Dawn will specify the properties of each asteroid and provide important clues about the early Solar System.**

**The Dawn project, as a part of NASA's Discovery Program of competitively selected missions, is managed and will be operated by the Jet Propulsion Laboratory (JPL) for NASA. The ion-propelled spacecraft is in development at Orbital Sciences Corporation (Dulles, Virginia), and the science instruments were developed and built by Max-Planck Institute (Katlenburg-Lindau, Germany), Agenzia Spaziale Italiana (Rome, Italy), and Los Alamos National Laboratory (Los Alamos, New Mexico).**

**When there is spacecraft collaboration between several industry partners, there is an inherent difference in integration and test (I&T) methodologies, which creates a challenge for verifying flight systems during the development phase. To converge the differing I&T methodologies, considerations were required for multiple project areas such as Flight System Testbed (FST), Assembly, Test, and Launch Operations (ATLO), and Spacecraft Simulator environments. This paper details the challenges and approaches of the JPL's effort in engineering a solution to testing the flight system with the Mission Operations Ground System while maintaining the comparability with testing methods of the industry partners.**

## Nomenclature

<i>ATLO</i>	= <i>Assembly, Test, and Launch Operations</i>
<i>ECP</i>	= <i>Extensible Channel Processing</i>
<i>FST</i>	= <i>Flight System Testbed</i>
<i>GDS</i>	= <i>Ground Data System</i>
<i>GUI</i>	= <i>Graphical User Interface</i>
<i>JPL</i>	= <i>Jet Propulsion Laboratory</i>
<i>I&amp;T</i>	= <i>Integration and Test</i>
<i>NASA</i>	= <i>National Aeronautics and Space Administration</i>

## I. Introduction

**T**he Dawn project's Integration and Test (I&T) effort included: (1) two Flight System Testbeds at Orbital Science Corporation to test flight software and databases, flight sequences, thread tests, verifying scripts and procedures prior to spacecraft use, (2) ATLO at Orbital Science Corporation to verify flight system including science instruments with the Mission Operational Ground System at Jet Propulsion Laboratory, and (3) four Spacecraft Simulator environments to verify science instrument functionality, flight software and databases prior to delivery. The challenges for integrating flight systems were perceived to be difficult due to various industry partners' diverse integration methods and locations, domestic and international.

---

<sup>1</sup> Senior Software Engineer, Mission GDS Integration Test and Deployment, 264-235

<sup>2</sup> Operations Engineering Program Element Manager, Exploration Systems System Engineering, 301-490

To ensure a high degree of mission reliability, JPL has developed a set of flight project design principles. One of those principles is “test as you fly.” To fulfill this “test as you fly” requirement, it became imperative for the Dawn Ground Data System to engineer a solution to allow for flight system testing with the Mission Operations Ground System while maintaining the comparability with industry partners’ testing methods. Moreover, since the Dawn project is one of NASA’s Discovery missions, where cost is a big constraint, a decision was made to take advantage of employing agile software development methods in developing a tool to satisfy the time and funding challenges. Thus, TCAssist, Test Conductor Assistant, was envisioned and developed as part of the JPL Dawn Mission Operations Ground Data System. TCAssist allows testers from different project areas to create test scripts and procedures that can be used within the Ground Data System for flight verification. Through use in the field, TCAssist allows testers to compare results from their specific ground support equipment with results from the Ground System. Additionally, the tool contains some dynamic features for command generation, database selection, and telemetry checking, among others. TCAssist was envisioned to be instrumental in implementing a unified approach for flight system verification for the Dawn project.

## II. Requirements

To facilitate flight system testing with the Mission Operations Ground System while maintaining the comparability with industry partners’ testing methods, an integrated I&T solution that could be used by every partner involved was levied upon the JPL Dawn Ground Data system (GDS). The need was recognized for new software that must be integrated well within existing ground software and that operates in several environments. Moreover, this software must create a closed-loop scripting capability and closely tie telemetry and commanding, among other requirements including:

- Integrate sequence software (and command databases) to allow for dynamic command generation
- Contain a closed-loop mechanism between telemetry and commands
- Contain a simple set of scripting commands
- Contain a comprehensive activity-logging capability
- Be compatible to industry-partner databases and scripting
- Provide closed-loop scripting capabilities for flight/ground integration and regression testing

At level 3, the requirements consisted of one, which was related to the verification of the flight system using a closed-loop scripting mechanism. The detailed lower level requirements were derived over the design and development period based on I&T experiences. Other requirements arrived late in the development cycle, which was challenged by limited time and resources.

### A. Environments

The Dawn integration environments included several different sites with differing characteristics. Four of the sites were for science instruments development, two were for flight system testbeds, and one was for spacecraft assembly. Each of the instrument sites had unique characteristics in I&T methods. Meanwhile, the testbeds and ATLO were similar, were co-located, and were operated by the same industry partner.

#### 1. *Spacecraft Simulator*

For the Science Instrument development, the JPL GDS was interfaced with a spacecraft simulator, as shown in Figure 1, uniquely tailored for each instrument developer. This setup was used to verify the instrument functionality with the Dawn Mission Operations Ground System: telemetry and command databases delivery, functionality validation of the interface with flight system, correct telemetry content validation with the system during mission operations, and correct command structure and functionality validation.

The Instrument developers used their preferred integration and test software to develop and verify functionality of their instruments in early phases. However, prior to their instruments delivery, the developing centers were to integrate their science instruments against the delivered simulator environment for verification with the flight and ground system. Additionally, each was to deliver a set of scripts and procedures to be used in the project I&T environment when the delivered instrument was to be integrated with the spacecraft. Then, these scripts and procedures would be used when the entire flight system undergoes sets of environmental tests.

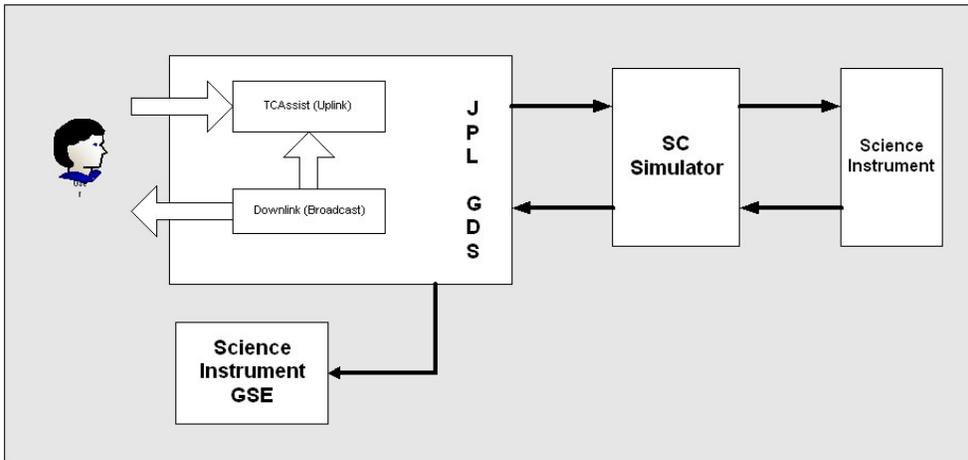
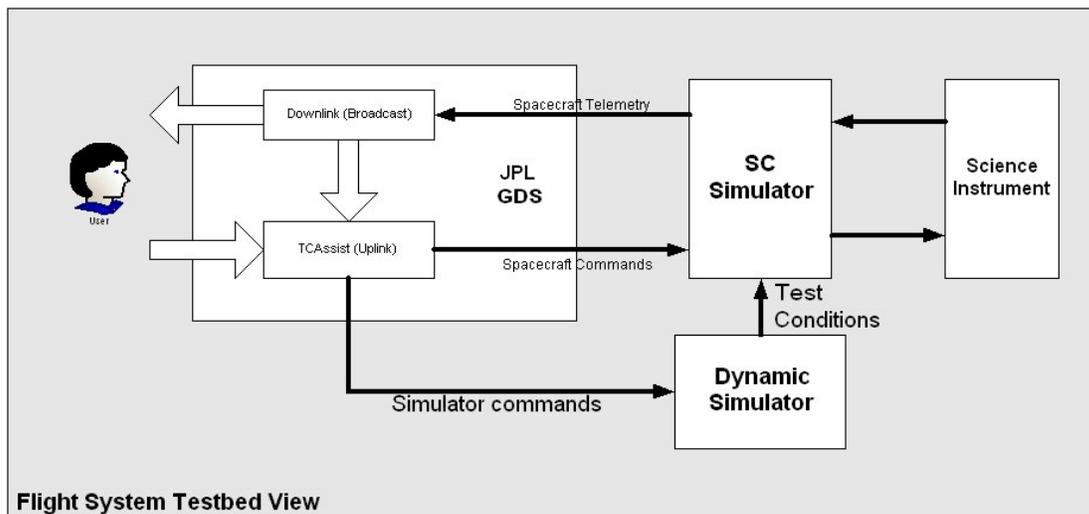


Figure 1. Typical interface of JPL Ground Data System with spacecraft simulator.

### 2. Flight System Testbed

Although the flight system testbed was mostly developed and managed by the Orbital Sciences Corporation for the majority of the development phase and co-located with the spacecraft I&T, it is required to be operated remotely by the JPL spacecraft team, via the a secure network, or by the Orbital Ground System. The flight system testbed consists of a fully functional flight system and the JPL Mission Operations Ground Data System. It also contains a software module for injecting space flight conditions for various flight subsystems. Figure 2 displays the I&T configuration used in the Flight System Testbeds and ATLO for flight system verification using the Mission Operations Ground Data System.



Flight System Testbed View

Figure 2. Flight System Testbed and ATLO I&T Configuration

### 3. Spacecraft Integration and Test

Spacecraft I&T contained several phases, such as subsystem integration, science instrument integration and comprehensive checkout, environmental testing, mission scenario testing, and launch. Many of these phases required the use of the Mission Operations Ground Data Systems, but generally speaking, most of the testing was performed by the industry partner with their own ground system. Although the spacecraft was developed by the industry partner, it will be operated by JPL during the mission; therefore, a verification of the flight system will be conducted using JPL data system as a fulfillment of the “test as you fly” requirement.

## **B. Closed-Loop System**

Initially, the requirements levied upon the JPL Ground Data Systems for meeting integrated testing methods between JPL and industry partners simply stated that there shall be a closed scripting environment. However, the further derivation showed that the lower-level requirements were numerous and detailed. The categories of requirements are listed as:

- Contain a simple set of directives
- Integrate seamlessly with the current GDS environment
- Dynamically link to command and telemetry databases
- Be capable of using different database versions
- Perform telemetry check by reading different connection types
- Provide detailed logging of activities and products
- Be configurable to work in the three different environments described above
- Maintain user controllability over script execution
- Allow user navigation through various modes

## **III. Interfaces**

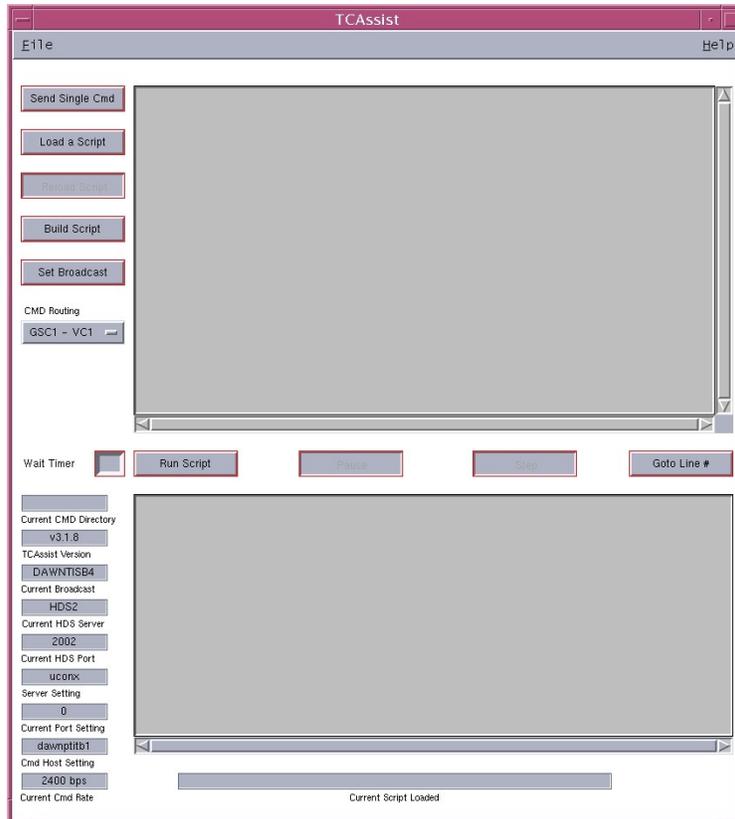
Based on the requirements and considering the current JPL Mission Operations Ground Data Systems software, the interfaces were defined to be:

- Graphical user interface (GUI) – Allows user to control software
- RS422 hardware interfaces - For I&T environments (via command subsystem)
- Socket interfaces – For testing with spacecraft simulators
- Sequencing subsystems – To integrate command generation software
- Telemetry processing subsystem – To access telemetry measurements

TCAssist, in essence, became a tool to tie together several subsystems within the Ground Data System, as well as add several unique features, to create an environment for the user in the I&T areas.

For the GUI, one main window was created to allow the test conductor to run and control a procedure script. (See Figure 3.) Within it, there is a display for the current script run, a highlight to indicate the script step currently executed, and a log area to indicate the results of the previous steps. Additionally, there exist several control buttons (such as pause and jump) to allow the user to control the flow. There are also several indicators for the configuration of the receivers and directories where events are logged. Many of the controls are available based on the mode the tool is in as defined by the mode navigation tree in Figure 4.

The GUI also includes a mechanism for users to generate and send immediate commands external to the script, thereby deviating from procedural script. The GUI contains a comprehensive list of the command database and associated parameters, with parameter limit checking. This mechanism leads to the interface with the sequencing subsystem. A command would be generated based on user inputs.



**Figure 3. TCAssist Main Window**

#### **IV. Implementation**

The implementation phase of the TCAssist software was started relatively quickly due to constraints on schedule, resources and requirements. A brief study was conducted to find an implementation approach, and an agile software development method was exercised.

Considering I&T requirements for the environments, capabilities, interfaces, and software need date, the decision was made that TCAssist development rely on software reuse for some components from the current JPL GDS, specifically those capabilities that related to the telemetry checking.

##### **A. TCAssist Software Components**

Based on requirements and interfaces defined, the following components were identified for implementation:

- Main Graphical User Interface – Allows users to control software
- Script builder – Allows user to create a script, integrating database commands and parameters
- Command builder – Generates the command into format for transmission
- Command database parser – Translates the database into human readable options
- Telemetry database parser – Translates database into human readable measurement identifiers
- Telemetry processor – Accesses the spacecraft telemetered measurements
- Script parser – Loads and parses script for valid syntax
- Script executor – Runs each script command as requested
- Environment configuration setup – Allows users to select configuration based on environment
- External interface executors – Executes commands external to software

## **B. TCAssist Directives**

A set of simple directives was agreed upon and implemented within the script executor and script builder of the TCAssist. The user is allowed several different types of script directives to create a test script. The directives are required to work together to create a closed-loop environment between telemetry and commanding. The set of directives is a comprehensive set to allow users in multiple environments to create scripts that are compatible between the environments specified above.

The contained directive types:

- Informative – Allows user to view information, such as:
  - Comments – User can add comments to script for information
  - Echo statements or values – Print out values or statement during execution
- Script control
  - Pause – Halt execution for a specified time period
  - Jump – Go to another script line
  - Variable settings – set a variable for use in other script portions
  - Logic statements – Allows user to use logical blocks for script execution
  - Looped execution – executes commands several times over
  - Nested execution – Perform logic in nested fashion
- Flight system commanding
  - Create and send a command in various formats (verbose or non-verbose)
  - Send a previously built command or sequence
  - Upload a binary file
- Telemetry checking
  - Echo a telemetry value
  - Halt the script until telemetry value is reached
  - Verify a telemetry value with tolerances

While each directive type has a specific function, they can be used together to create a multifunction. For example, a telemetry check can be used with a control statement, or a looping variable can be used within as a command parameter.

## **C. Telemetry Processing Component**

The telemetry processing component relied heavily on software reuse of the existing JPL GDS Java components, called ECP (extensible channel processing). Through the use of ECP, the telemetry values were easily accessible in many forms within a script. A telemetry measurement was available in raw form (engineering value or state format), which was also dependent on the definition within the telemetry database.

Upon TCAssist startup, the telemetry processing module is initiated and waits for requests. As can be seen from Figure 3, when a telemetry measurement is requested, the scripting engine issues a client request, which is handled by the server. The server listens to the telemetry stream for the specified telemetry measurement. Once read, the value is returned to the client, which in turn formats the value and returns to the scripting engine. The multi-threaded client request handling allows the user interface to maintain controllability during the telemetry searching.

## **D. User Control and Navigation**

TCAssist is a tool to be used in many different I&T environments; and therefore, it required user control and navigation of scripts. The development of a script is usually a lengthy iterative process. Users create the first draft; run within the given environment; and usually encounter errors in the logic, script sequence, or input parameters. They then must repair the errors, reload, and try again. These iterations continue until a desired script is complete. The iterations require the user to have control and the ability to navigate within the script. This entails activities such as skipping a line, jumping sections, returning to previous sections, reloading a script, changing configuration, and many others. Therefore, it was well understood that the user required adequate navigation of modes.

Figure 4 shows how a user can navigate from one mode to another, as well as the limitations. Within any particular script run, a user can execute step-by-step, run through the rest of the script, pause, cancel, jump to a line, among others.

## E. Performance

The performance requirements against the TCAssist were not defined when implementation was initiated. However, the following three areas were discovered to be key for performance implication: the script executor; the telemetry processor; and the command executor.

The script executor performance was dependent on the implementation language and was observed to meet user requirements for script execution. However, it was discovered mid-way through implementation that the script parser was actually a performance bottleneck in certain situations. Each script command needed to be translated into an execution command from a user script entry. When enormous scripts were generated, the parsing took longer than desired. The delay in translation and script navigation was too slow to adequately meet the user controllability requirements. This was mitigated by performing the script translation in a streamlined fashion at the script load, rather than at script execution.

The command creation and sending component was wholly dependent on external tools and interfaces within the sequencing and command subsystems. It was observed that the sequence generation was slow, although there were no time/performance requirements. If several single command sequences were generated repeatedly, many steps were unnecessarily repeated. Based on the observations, two independent enhancements were implemented to mitigate the command execution. First, a non-sequencing external component was created to generate the commands, providing a 130% time enhancement. However, this was not the preferred solution, as it bypassed required sequencing software. The second enhancement involved a sequence generator enhancement removing the repetition of unnecessary steps, which showed a 125% improvement. This was acceptable for the given resources.

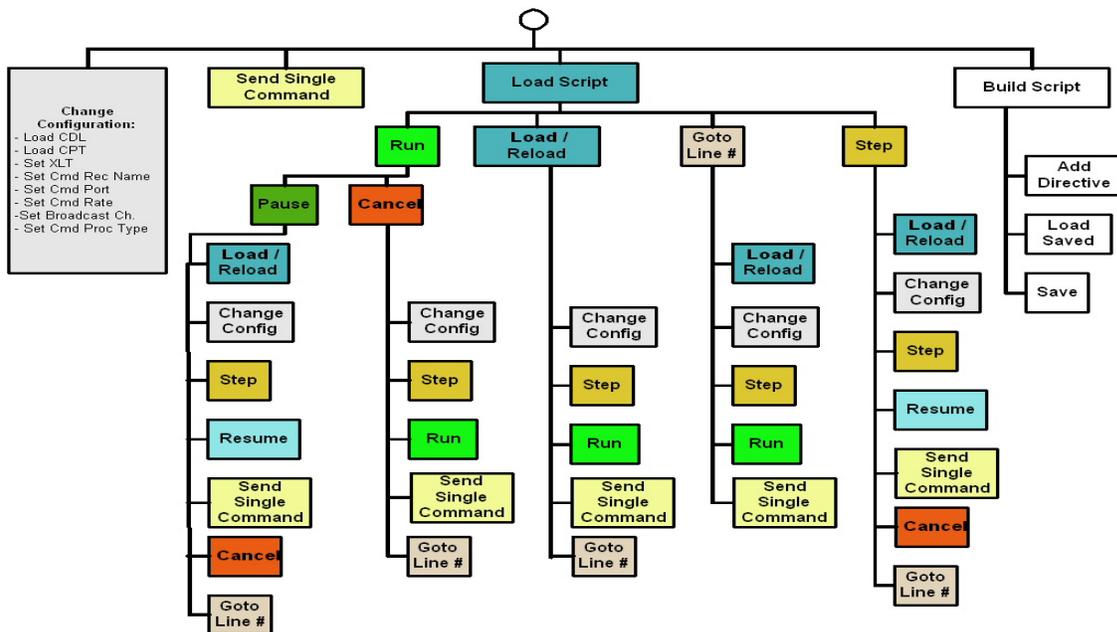


Figure 4. Navigation between modes.

For the telemetry processor, there was any performance requirements defined initially. Throughout the implementation and testing, it was observed that the highest telemetry rates flooded the stream with bursts of data. At times, a non-repeating measurement could be missed, although not frequently. The telemetry measurement reading reliability was at stake. Therefore, the telemetry processor was modified to filter the stream and search for desired measurements only. It is known that most user scripts search for a few measurements at any single point, so the telemetry processing server within TCAssist was modified to filter for desired telemetry measurements. This method proved to be reliable and acceptable within the given resources.

## V. Lessons Learned, Experiences, and Rationale

The overall development spanned more than one year in several phases, limited by the two major forces of time and funding, from which many experiences and lessons stemmed. From the time we learned of the single overarching requirement and delivery of version 1, there was approximately 1 month. During this time, system engineering conducted a limited study for implementation and mitigation strategies, the developers then started to code the initial version, which was intended to integrate command database, sequence software interfaces, and hardware interfaces.

The low-level requirements were not well defined and had to be collected during the development phase. This caused the developer to take implementation paths that sometimes required re-work. It was known that requirements must be defined fully from the beginning, but the users and participants were not available early on in the project cycle. Also, software usage was light in the initial stages and increased slowly.

A proper test environment is necessary for full validation; however, limited hardware resources and funding curtailed the ability to obtain a proper test environment for full validation. Additionally, the performance requirements did not exist. This led to a couple of performance issues discovered with high data rates that required resolution in later phases, discussed in sections above.

Implementation of dynamic command generation was a new concept in the general integration of the JPL AMMOS software and proved to be very useful for script generation. A user can access the command database to view available commands and associated parameters and limits. This allows a user to verify a command or send a command to the flight system without having to verify it outside the system.

The implemented script builder can be a handy feature that allows users to enter commands and associated parameters for the initiation of a script.

There are numerous future enhancements that can be added to the TCAssist software to make it more useful, such as:

- Enhanced log file viewer to allow users to clearly see the activities in a particular test. This viewer could help system engineers and interested parties easily parse through activities in a test.
- The script control directives can use more logic constructs and controls. The contained set is sufficient for the conceived scenarios, but more constructs could make for more efficient scripting.
- Create features that allow for more dialogues between script author and script executor, such as a dialogue that tells tester to perform a step before continuing.

### Acknowledgments

Many thanks to JPL Colleague and the TCAssist software developer Lamar Gilliam.

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