A Multi-mission Event-Driven Component-Based System for Support of Flight Software Development, ATLO, and Operations first used by the Mars Science Laboratory (MSL) Project

Navid Dehghani
Michael Tankenson
California Institute Of Technology
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109-8099

ABSTRACT

This paper details an architectural description of the Mission Data Processing and Control System (MPCS), an event-driven, multi-mission ground data processing components providing uplink, downlink, and data management capabilities which will support the Mars Science Laboratory (MSL) project as its first target mission.

MPCS is developed based on a set of small reusable components, implemented in Java, each designed with a specific function and well-defined interfaces. An industry standard messaging bus is used to transfer information among system components. Components generate standard messages which are used to capture system information, as well as triggers to support the event-driven architecture of the system. Event-driven systems are highly desirable for processing high-rate telemetry (science and engineering) data, and for supporting automation for many mission operations processes.

Additional components of the system, including an archive and catalog service, provide long-term archival for science and engineering products, as well as the metadata associated with them. Data accountability is based on access to the catalog of messages and an accompanying information model, adaptable for each subsequent project, to track the flow of data through the system and to provide end-to-end accountability.

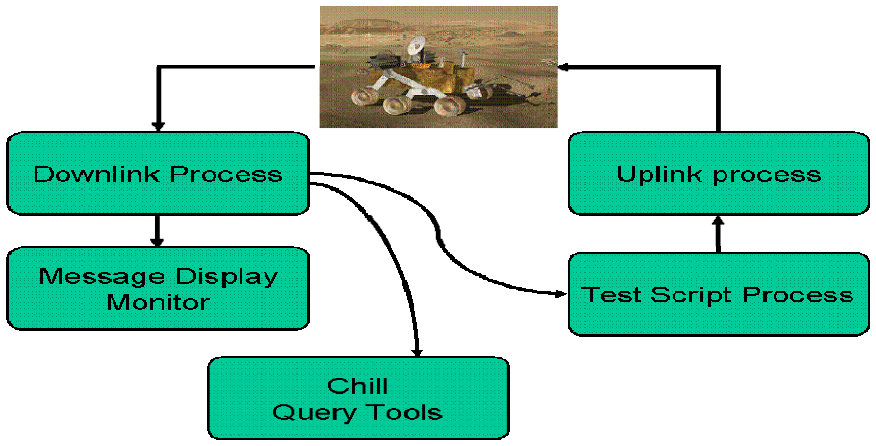
By capturing all of the data, metadata, and messages for the system in an accessible catalog, it becomes possible to create automated uplink/downlink activity reports and other operations products. Mission Operations processes can be automated, with a significant reduction in operations costs.

INTRODUCTION

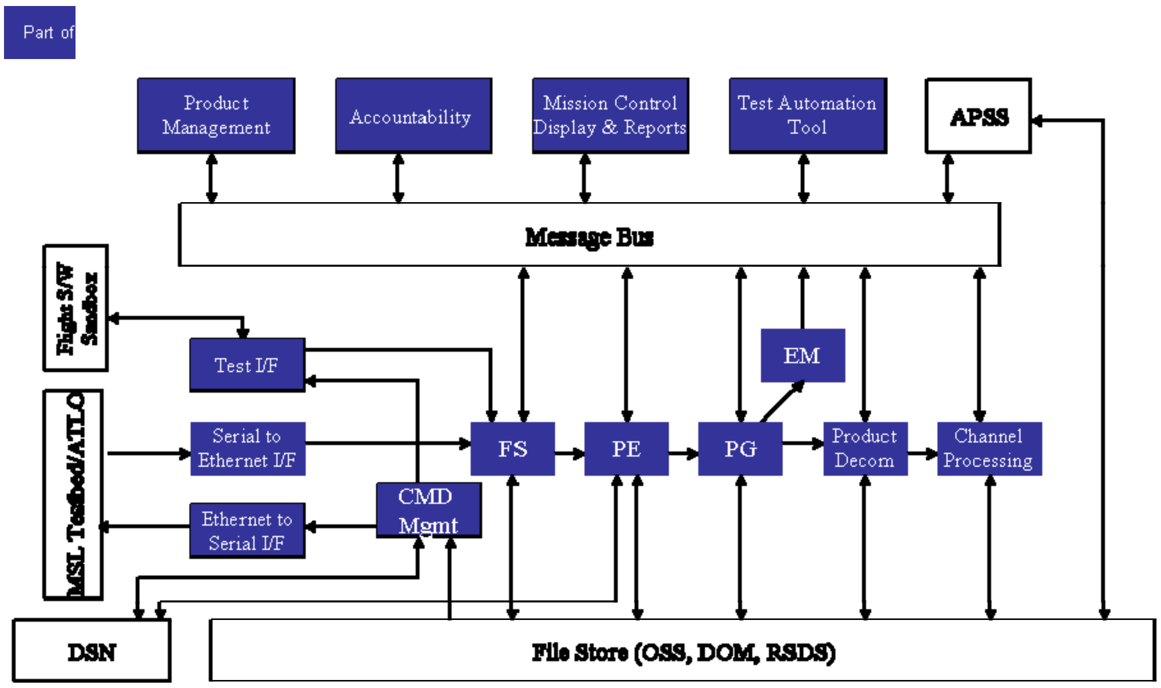
In August 2005 the MSL project adopted developing a Ground Data System (GDS) that would 1) support all mission phases of the system implementation and operations with a common system (e.g. a GDS supporting the early stages of flight software development, project's various high fidelity testbeds, ALTO and supporting cruise and surface operations), 2) have a common Linux based platform for use during all mission phases and with all remote scientists and collaborating entities, 3) to reduce the risk of developing a new flight system, including flight software (FSW) and GDS for MSL, have strong inheritance from the Mars Exploration Rover (MER) project as well as other successful GDS and FSW implementations, 4) be built on modern day standards and technologies (e.g. Event Driven, Java, XML, service oriented architectures), thus ensuring a highly interoperable system that is rapidly built and is responsive to the inevitable requirement changes that creep into every system development as the system implementation progresses, and 5) be compatible with emerging Deep Space Mission Systems (DSMS) architectural standards, thus ensuring long term interoperability with the DSN and next generation DSMS architecture. In October 2005, the JPL Multi-Mission Ground System and Support (MGSS) Program office adopted this approach as the basis for the next generation multi-mission ground data system and provided additional support in partnership with the MSL project. This capability was later named Mission Data Processing and Control System (MPCS)

MPCS is developed using the Agile development methodology to form the core of the task's software development process. The Agile development methodology enables the system to deliver a product that meets user needs earlier than with more tradition methods. To meet the MSL project needs the system is developed in five phases, each phase is built on the capabilities of the previous phase and incorporates lessons learned from its usage. Phase 1 is an early prototype for proof of concept. Phase 2 is the flight software developer sandbox. Phase 3 system is developed to support various MSL testbeds. Phase 4 system provides ATLO support. Phase 5, final phase for MSL, is for operational support. Key tenets of Agile development process are described in the next section.

At the top level, the system is comprised of five major components; a core uplink process for processing and managing spacecraft command data, a core downlink process for processing and archiving spacecraft science and engineering telemetry data, a distributed message display and monitor process, a test scripting process, and a set of report generation tools for archived data.



A major architectural tenet of the system is that: major processes (functions) communicate via standard messages through the JMS message bus. This approach to system design decouples applications and allows for a “plug and play” architecture. Using an industry standard message bus provides guaranteed critical uplink and downlink data delivery. Another architectural tenet of the system is the event-driven system design. An event is communicated using standard messages which are subscribed to by other processes within the system. In this type of architecture standard message definition and subscription strategy defines how a system does its processing. This system design can easily be augmented to produce accountability information/messages as part of its normal workflow. An accountability service (plug in) can subscribe to accountability type messages and, using an information model tailored to each mission, provide the necessary accountability function. MPCS major components (highlighted in blue) are shown in the diagram below. As it is shown all the components are used during flight software development, during ATLO, and operations fully complying with the test as you fly paradigm. Compatibility with DSN is assured during ATLO and toward end of ATLO with DSN Compatibility Test Trailer (CTT).



The MPCS will be initially developed to support MSL mission requirements for the telemetry and command components of the MSL Ground Data System (GDS). During the early phases (phases 1 and 2) of the MPCS implementation, multi-mission features will be designed into the system, to the extent possible. The MPCS task is programmatically managed by the Mission Control, Data Management and Accountability, and Spacecraft Analysis (MDAS) organization within the MGSS program office. Further sections in this paper details the architecture of the MPCS, the messaging design, the downlink process, and the uplink process, followed by a discussion of the “test as you fly” paradigm.

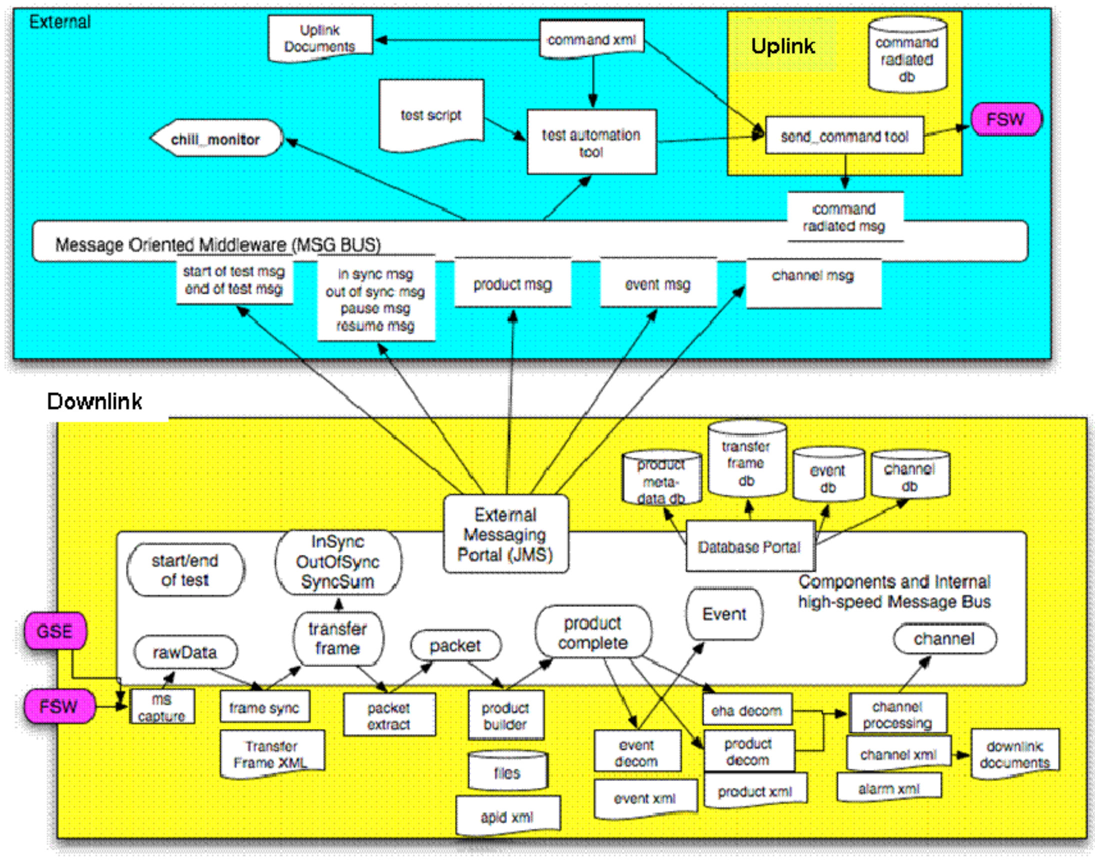
ARCHITECTURE

MPCS has strong inheritance from GDS components (e.g. Product Builder, Channel Processor) that have been developed for other Flight Projects (MER, MRO, DAWN), and are currently being used in operations and ATLO. The MPCS component architecture described in this paper is derived from the event driven model used to support Mars Exploration Rover (MER). This architecture uses a JMS messaging capability (reliable data transfer) to trigger events and status which in turn enables “plug and play” architecture. The MPCS components are Java-based, platform independent, and are designed to consume and produce XML-formatted data. MPCS components are designed to be multi-mission, or easily adapted for future missions.

The MPCS architecture enables GDS/MOS processing to be driven by Spacecraft and Ground events, rather than real-time telemetry monitoring. This is done by generating standard messages (events) and sending them on a JMS Information Bus. Information is represented at a high semantic level in reports, products, and messages on both spacecraft and ground enabling operators to plan based on activity and performance reports. This leads to more efficient mission operations, more automation, and better science return.

For the purpose of discussion in this paper two type of messages are defined; event/informational messages, and data messages. A data message is defined as a message containing, as an example, spacecraft science and engineering data. An event or information message is defined as a message containing information about the processing of spacecraft science and engineering data. Data messages by nature have a much higher data volume than event or informational messages and in some applications (e.g. high data rate missions) it is not advisable to publish both types of messages on a single “public” message bus. Trade analysis for supporting each mission is key in determining the best design. The architecture of the system should allow for easy configurability in terms of availability of type of messages to the public information bus. The MPCS architecture uses a message portal design for configuring the availability of messages to the “public” and “internal” message bus in the deployed state.

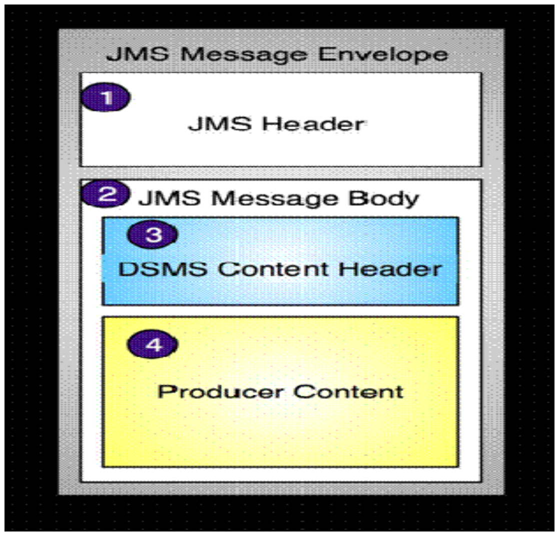
A detailed architecture of the MPCS showing both the “public” and “internal” message bus is shown below. Both uplink and downlink components are shown. The downlink processing is essentially a pipeline processing of spacecraft science and engineering data through various processing stages from frame synchronization to channel processing. An external messaging portal is deployed for publishing messages to the public message bus. Published messages can be subscribed to by any other component or future plug-in to the public message bus. The external message portal is configurable so that if a message is determined to be used by a subscriber can be published to the public message bus.



Designing an event-driven operations system requires reliable data delivery. Ground systems can not miss critical mission events which are communicated using messages. Messaging architecture therefore forms the basis of such a design and is further specified in this paper.

Messaging

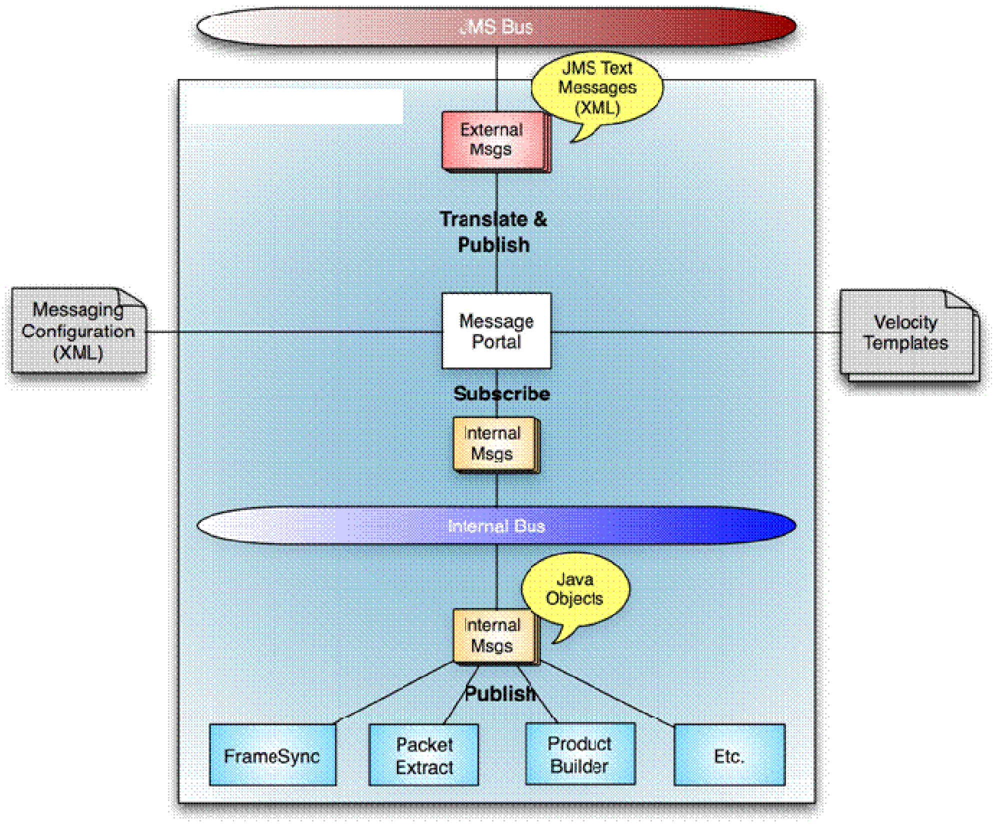
A set of high level requirements were defined early in the analysis phase of the MPCS. First, internal messages deemed to be of use to other applications must be published to the external JMS bus by MPCS components. Second, which messages are published and how often they are published must be configurable. And third, subscribers of messages shall be able to filter by at least, Mission, Spacecraft, Message Type, and Test ID. A message consists of a JMS header and a JMS message body in ASCII format. The JMS message body consists of a standard DSMS header followed by internal message content which is expressed in XML format. All standard message formats are defined in Velocity, a Java based template system that provides an easy to use, high-level language for conversion of Java objects (internal messages) to XML format JMS messages. Message type is the primary identification of messages throughout the system and provides the basis of subscription by any other plug-in component.



MPCS Downlink Process

As described earlier, the downlink process is essentially a pipeline processing architecture. In this type of architecture each processing component (frame synchronizer, packet extractor, product builder, etc.) performs a specific action on the downlink data preparing it for further processing. Each component of the downlink process is a pluggable component to the message bus (in this case internal message bus). Other components can be developed and plugged into the downlink process without disturbing the overall architecture as long as it fits within the standard messaging definition. As an example, if a better implementation of packet extractor is developed at a later time (or for a different mission), as long as it produces standard messages that can be interpreted by the subscriber of the packet extractor messages, it can be "plugged in". One note of caution about this type of architecture is that one can not indiscriminately plug in or plug out processing components at random. There is an inherent limitation to this type of architecture in that if a new component is developed that produces a type of a message that is not recognized by any other components of the system, it can not be plugged in.

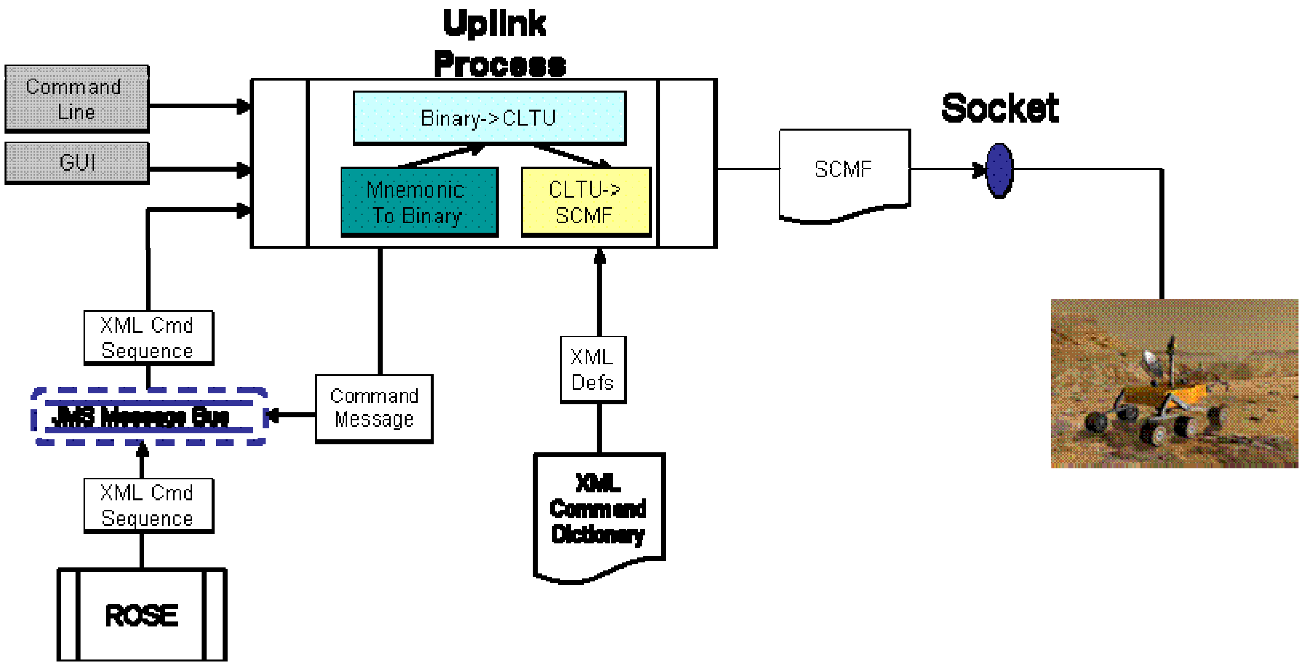
As seen in the diagram below an internal message bus is used to transfer high data rate information among these processing components. A message portal converts internal messages (that are subscribed to by any other plug-in component) to XML format and provides those messages to the public message bus. The easily configurable system allows for adding or deleting any message type to the public message bus. As an example, all accountability type messages subscribed to by the accountability service is configured and sent to the public information bus.



MPCS Uplink Process

The MPCS uplink process subscribes to command sequences messages. Command sequence messages are generated by the MSL sequencing system and send to the JMS (public) message bus. Upon receipt of these sequence messages, the MPCS uplink process used the command dictionary in XML format to convert sequences to binary commands and formats them into a Spacecraft Command File (SCMF). SCMFs are products that can then be sent to the spacecraft by DSN radiation service. During testbed and ATLO phases a serial connection provides the needed connectivity to the flight articles. The processing components of the MPCS Uplink Process perform the following functions:

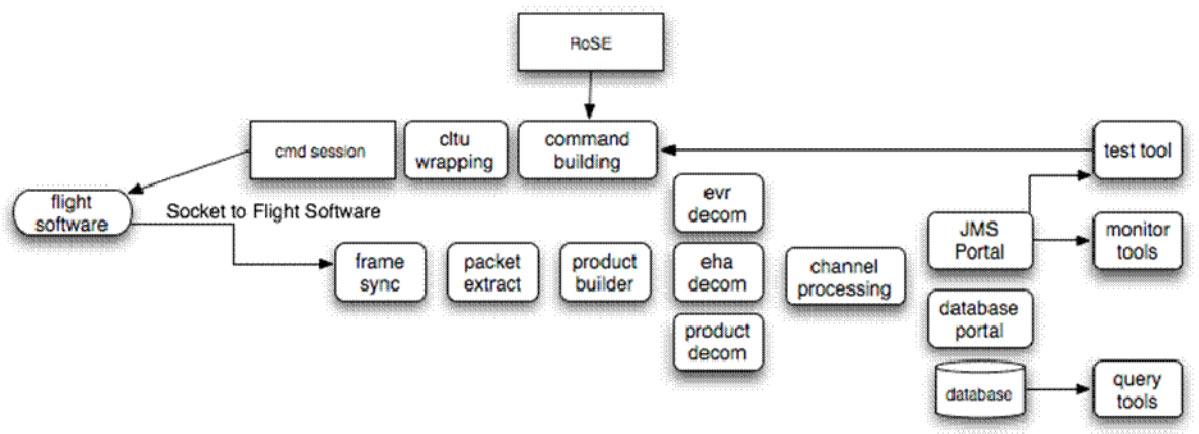
- Mnemonic to Binary translation. This component subscribes to command sequence messages generated by the MSL sequencing system and using the XML command dictionary generates the appropriate binary information.
- Binary to CLTU component formats the command binary information to Command Link Transmission Units.
- CLTU to SCMF component packs the CLTUs into a file of binary spacecraft commands suitable for transmission to spacecraft.
- Each component generates the appropriate messages that are sent to the JMS or public message bus.



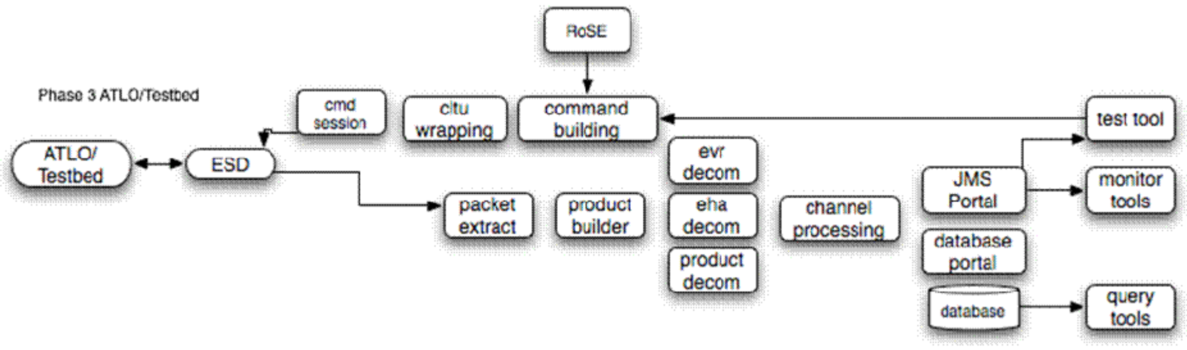
Flight software development to operations (Test as you Fly paradigm)

A major strength of this architecture is its flexibility to support all the development and operational phases of a flight project. Components developed and used for one phase are evolved and used for the subsequent phases. Major component are then used from early flight software developer sandbox to testbed, ATLO, and eventually operations, fully supporting the “test as you fly” concept.

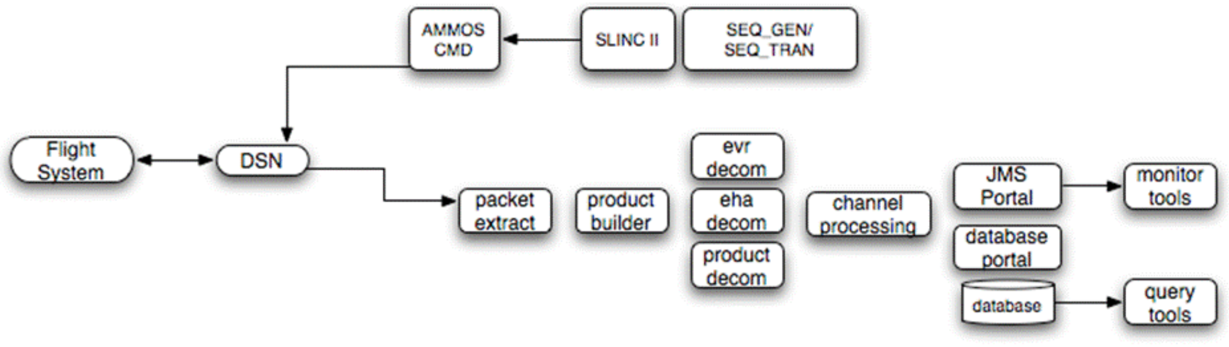
For the flight software developer sandbox, an implementation of the MPCS software components is developed and deployed along with flight software development environment to about 30 flight software sandbox environments on Linux workstations. Uplink and downlink components of the MPCS for the flight software developer support are shown below.



MPCS components for testbed and ATLO support are shown below. In this phase serial interface to testbed and spacecraft is provided using a mission space board that performs frame synchronization, Reed Solomon decoding, and turbo extraction. During ATLO periodic tests with DSN ensures DSN compatibility. This is culminated by testing with the DSN compatibility test trailer which is usually available toward the end of each project’s ATLO phase. MPCS capabilities include one downlink, one uplink, one test process per data stream, and 10-20 workstations for message display monitor process along with query tools will be provided.



During Operations one MPCS downlink process for each data stream, one MPCS downlink process for relay processing, and up to 200 workstations with MPCS monitor processes and query tools will be deployed. Sequence generation, translation and uplink (Command) is used to generate, translate, and send commands to spacecraft using the DSN radiation service. Remote science operations interface to MPCS at the time of writing this paper has not yet been designed.



Agile Development Process

The MPCS has adopted many key tenets of a new class of software development processes commonly referred to as the Agile Development Process. While the MPCS task does not fully conform to all of the Agile development practices, it does follow key practices which we believe enable products to be delivered more rapidly and with higher customer satisfaction than with more traditional methods. The Agile development practices are a set of interdependent practices that ensure the development process and products respond well to change – a common problem with traditional methods. Implementation team members follow the practices described below and these practices are an integrate part of the team’s software development processes. A quick snapshot of the Agile development practices adopted by the MPCS task follows:

- Task Planning: During each phase’s planning, quickly determine the scope and content of the next release by combining business priorities and developer's estimates of implementation effort. Grass roots estimates based on prior experience are encouraged.

As development experience and data on actual implementation effort grow, increasingly accurate effort estimates are possible. As the content of future releases changes, based on user experience with prior releases, the development plan is updated. Note that the plan for each release fits within a larger plan that may span several months or multiple years, and that development of each release is done via fixed time period development cycles.

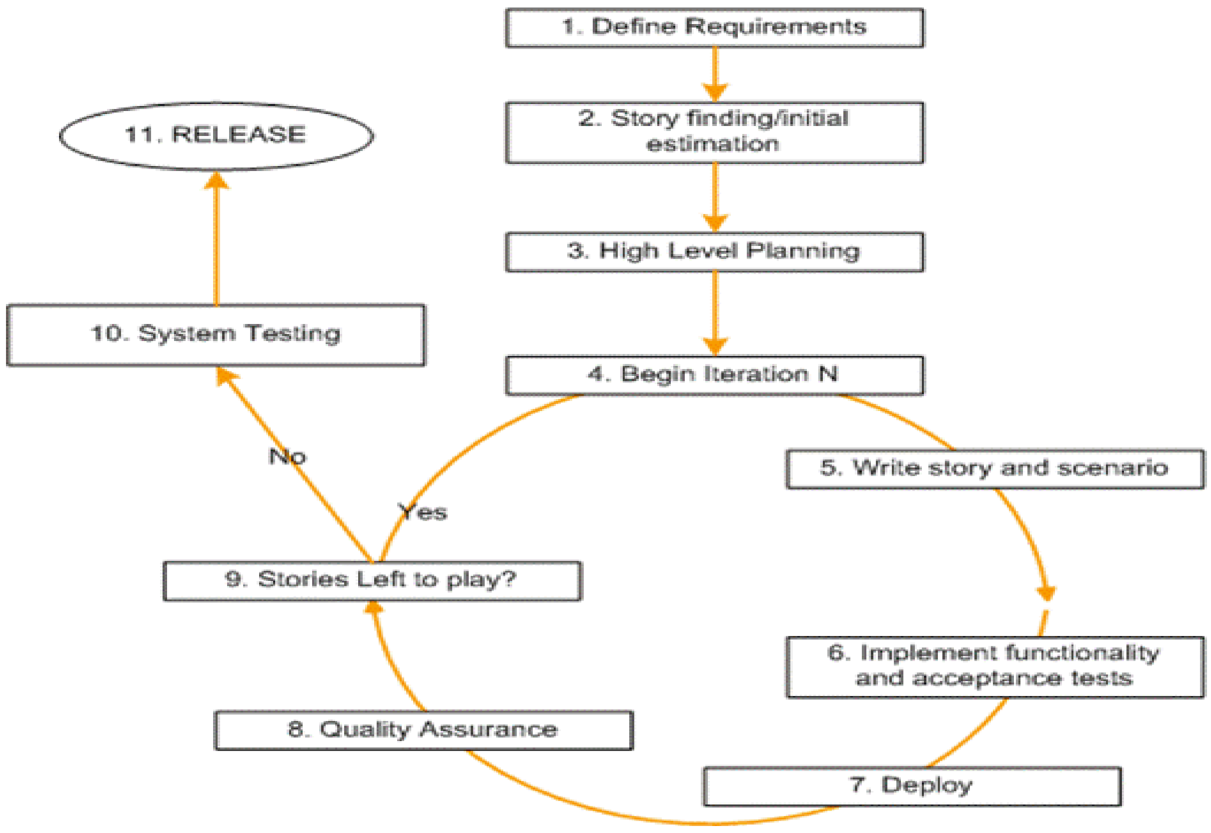
- Small Releases within a fixed time period development cycle. Each development cycle is time fixed to a number of weeks that initially results putting a simple version of the system into operation quickly to permit user reaction and to firm up recommendations for future releases. Each subsequent release builds upon the previous release and is available to gather additional user feedback. The MPCS task has selected 4 week development cycles.
- Simple Design: The MPCS should be designed as simply as possible to meet known requirements. Designs that anticipate "future requirements" is strongly discouraged. Unnecessary complexity is removed continually by refactoring/restructuring the system without changing its behavior (as opposed to developing new capabilities from scratch). The continually growing set of unit and system tests delivered with each delivered unit of code are used to ensure that system behavior has not changed.
- On-site Customer Participation. Include a customer representative on the team to clarify requirements, validate designs and implemented functionality, define new requirements and provide feedback to each new release. The appropriate MPCS users for each phase attend the weekly design team meetings.
- Test-Driven Development (TDD). In Test-Driven Development, a developer states their intention in the form of a story and set of tests, before implementing code. This process tends to make their intention as simple and readable as possible. TDD has several powerful effects. First, every function of the software will have tests that verify its operation. This allows developers to attempt code changes with a great deal of confidence. Second, it causes developers to focus immediately on class _interfaces_ (as well as functions), resulting in software that is easily callable. Third, it forces developers to write testable software. Software that is both testable and callable is decoupled - a perennial design best practice. Additionally, the tests themselves serve as valuable documentation - new developers can view them to understand how to call and work with unfamiliar classes. Some key points to remember about TDD:
    - Developers do not have to write a test for every single method they write - only production methods that could possibly break. Developers are free to do

exploratory coding themselves from time to time, but when it is time to write code for production, they must throw it away and start over with unit tests.

- Under test Driven development, Unit Tests are more than Unit Tests that are written to test that the code written works. MPCS Unit Tests are written to define what it means for the code to work.
- Using Test-Driven Development implies that there is always an exhaustive test suite available to test the current code baseline. This is because there is no code unless there is a test that requires it in order to pass. Developers write the test, then (and not until then) write the code that is tested by the test.
- No code goes into production unless it has associated tests. Changes are not made to the code without some way to confidently tell whether the developer has broken the behavior. A feature does not exist until there is a suite of tests to go with it. The reason for this is that everything in the system has to be testable as part of the safety net that gives the development team confidence and courage. This also gives the developer the confidence to make changes and refactor when necessary because the developer knows that the changes make will not break the code baseline as long as the unit tests are executed successfully.
- Write the tests first. When a developer has a task to do (some bit of functionality to implement) they write code that will test that the functionality works as required before they implement the functionality itself.
- Tests determine what code developers need to write. By writing only the code required to pass the latest test, developers are putting a limit on the code they will write.

- Continuous Integration. Code is integrated and tested after a few hours - a day of development at most. Integrating one set of changes at a time works well because it is obvious who should fix a test that fails – the developer who wrote the change should since they must have broken it. Our Software Configuration Management approach using Cruise Control performs continuous integration and notification of build or test failures to the appropriate software developer in real time.

Agile development process as used for MPCS is shown below.



The Agile Release Process - Sam Newman
http://www.magpiebrain.com/archives/2005/02/14/release

SUMMARY

The MPCS is an event-driven, component-based system. It represents the next generation architecture for developing ground data processing components providing uplink, downlink, and data management capabilities for the Mars Science Laboratory (MSL) as its first target project. MPCS is developed based on a set of small reusable components, implemented in Java. Each component produces and publishes standard messages to JMS message bus which are received by any subscriber component on the system forming the basis of the architecture. This architecture allows for flexible design and incremental additions of new capabilities (components or services). As an example, an accountability service can be developed which can subscribe to all accountability type messages and use a mission's information model to automate its monitoring needs. The architecture of the MPCS enables automated mission operations which can significantly increase its efficiency and reduce its operational costs.

ACKNOWLEGEMENT

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.