AIAA-99-4454

SOFTWARE LIFECYCLE THEMES FOR JPL MISSION DATA SYSTEM (MDS) PROJECT

Anne B. Elson
Jet Propulsion Laboratory
California Institute of
Technology
Pasadena, Ca. 91109
aelson@jpl.nasa.gov

Abstract

Today there are many small deep space missions in progress or in conception at the Jet Propulsion Laboratory. These missions have short lead times and small budgets while still pursuing ambitious science and technology goals. The short development times, the small funding profiles and the overlapping schedules of these new missions preclude the intensive, one-of-a-kind software development, maintenance, and operations efforts that were possible during the era of the big missions like Galileo and Cassini. How the laboratory develops, maintains and operates mission software in this new environment of multiple, concurrent, better, faster, and cheaper (BFC) missions will be crucial to the success of these new missions. The Mission Data System Project (MDS) team is developing core mission data system software for a group of the new BFC missions. As part of this effort the MDS team is piloting a different approach to mission software development for the laboratory. This paper describes the MDS software lifecycle approach and the ways in which this approach differs from past mission software development efforts. Additionally this paper discusses the ways in which the MDS software development approach should contribute to the success of the MDS BFC customer missions.

MDS Overview

In April of 1998 the Jet Propulsion Laboratory (JPL) initiated the Mission Data System Project (MDS). This project has been chartered to rethink the entire mission software lifecycle for the types of deep space missions that JPL has traditionally designed, built and flown.

The MDS team has proposed the development of a unified flight, ground and test software data system for deep space missions. This software system will be component based and adaptable to a variety of current and future missions. The MDS is characterized by a number of architectural themes. These themes and how they contribute to the success of better, faster, cheaper (BFC) missions are discussed in detail in a related conference paper "Software Architectural Themes in the Mission Data System".[1] The MDS architectural approach can be summarized as follows:

The MDS software development approach makes a state based architecture the focal point of mission software analysis and design. This approach proceeds from two central architectural principles and their corollaries:

Subsystems are constructed from architectural elements – not the other way around

- Find the problems in common
- Create common solutions
- Tailor the general solutions to the particular problems

Managing interactions is the foundation of a design

- Find the interaction mechanisms (de-coupling where feasible)
- Otherwise, create coordination services for the interactions
- Control interactions through these common services rather than function-to-function

The MDS team is building a set of mission software frameworks based on these principles. This set of frameworks forms the core around which the rest of the MDS software will be built. The frameworks are constructed around a few basic notions familiar to spacecraft system design. The most important of these is the notion of "State". State is defined as a representation of the momentary condition of an evolving system and is a central organizing theme of the MDS architecture.

Models describe how a system's state evolves. State information and models together provide the user of a system with the information on how to operate that system, to determine or control its future, and to assess its performance. MDS developers are producing state determination and control software frameworks (patterns) that will be instantiated within each of the software application domains included within the MDS.

The MDS team will produce all of the standard flight, ground and test software capabilities that deep space mission customers typically need during mission development. The MDS team will also produce some mission operations software. However not all existing JPL ground system software is expected to be rewritten for the first MDS development effort. The MDS team will provide a seamless interface between any new MDS ground system software and those parts of the previously existing ground system that remain.

The MDS team will design and build a mission software system for a fictional reference mission. This mission's operational scenarios, spacecraft design, ground system capabilities, and test environment will look very similar to those of the first MDS customer mission, the Europa Orbiter (EO) Mission of the Outer Planets / Solar Probe (OP/SP) Project. The MDS reference mission will also contain features and capabilities that demonstrate MDS software flexibility and adaptability for future mission technologies and science opportunities other than those needed by the Europa Orbiter Mission. The Europa Orbiter customer will adapt MDS frameworks and application subsystem instantiations to their own mission needs. The Europa software team will use some pieces of the MDS reference mission software directly. The EO team will also take MDS application software frameworks used to generate the reference mission and adapt, extend and/or replace portions of them to create their own mission unique software system. In either case the customer mission is expected to maintain core MDS architectural themes that have been implemented as a set of software patterns throughout the MDS application frameworks.

The initial MDS software system will provide mission software capabilities that are equivalent to mission software capabilities of recent JPL deep space missions. As noted above however MDS software will also contain several new, and/or expanded mission software capabilities. One of the team's goals is to produce a software product that enables spacecraft for deep space missions to be both more autonomous and easier to operate than the mission software systems of previous missons. Autonomy concepts that were tested with the Deep Space 1 (DS-1) Remote Agent Experiment will be enhanced and extended within the MDS. The MDS team will build a system in which optical navigation products are produced on board the spacecraft and then used by it autonomously to compute new trajectories.

By providing a component based architecture within a unified flight ground system the MDS team is planning to produce a software system that is easily reconfigurable. A component based design should facilitate movement of software capabilities between the ground and flight systems of mission customers. Location of software capabilities such as spacecraft trajectory correction determination may depend upon the needs and constraints of a particular mission or upon phases of that mission. This capability could migrate from the ground to the spacecraft as a mission's needs and objectives changed over time.

MDS Software Lifecycle Approach

A lifecycle model provides a set of development guidelines to the developers. It identifies a set of development phases and the work products (artifacts) to be produced in each. The model helps to bring structure and standardization (predictability) to an activity that often appears to be chaotic and unpredictable. The MDS software lifecycle can be viewed as a set nested loops: one outer loop and an associated set of inner loops. The MDS project will traverse one cycle of its outer loop and multiple cycles of its inner loops to produce MDS flight, ground and test software for a mission customer.

In the MDS lifecycle the outer loop of the lifecycle maps to management and software system engineering activities. The outer loop divides into 4 phases: feasibility, elaboration, construction and transition. The MDS outer loop is primarily incremental but phases overlap somewhat. The outer loop includes systems requirements analysis, systems partitioning analysis (primarily hardware software and flight ground partitioning), object analysis, system architectural design, system validation and test, and system maintenance. Top level MDS requirements and systems analysis efforts need to be incremental to support mission customers and the X2000 1st Delivery Project. These projects must baseline top level system functional requirements early their development lifecycle in order to make timely hardware decisions. Many JPL deep space missions have fixed launch windows that if missed will not re-appear for months or years. Hardware design as well as hardware software trades need to be made early in a mission project's lifecycle and frozen so that the hardware can be built (or procured), assembled and launched on time. The MDS lifecycle approach with its incremental outer loop activities accommodates this need.

MDS inner loop activities and phases closely follow the software lifecycle model of the MDS Object Oriented Analysis and Design consultant, Bruce Douglass. Douglass promotes a spiral software development lifecycle model with iterative proto-typing.[2,3] In the Douglass lifecycle model software developers traverse a set of software lifecycle phases multiple times. During the earliest iterations of this lifecycle model developers implement a complete but thin version of the entire system. All major system interfaces are implemented but many if not most of the internals of the software components attached to these interfaces are only stubbed in. During each subsequent iteration of the lifecycle development teams increase the capabilities of groups of components. Each development cycle has a particular technical focus. With each iteration of a inner loop cycle work products (artifacts) grow in their completeness and quality until all agreed upon system requirements and constraints are achieved (or re-negotiated).

For each circuit of the MDS outer loop multiple MDS development teams working in parallel complete multiple iterations (cycles) of a set of associated inner loops. Each MDS inner loop represents one software domain (one area of technical expertise for mission software development) within the MDS. An inner loop is divided into 3 phases of software development: analysis and design, implementation, and evaluation and test. Once system interfaces are implemented in the earliest iterations of a inner loop cycle the development teams can proceed somewhat asynchronously to one another during subsequent inner cycles if they choose to do so. The MDS team expects to have planned, periodic alignments of inner loop completions across multiple software domains (sometimes referred to as synchronization points) throughout the lifecycle. These are currently planned to occur at six month intervals. These alignments provide for periodic re-integration of cross-domain capabilities. They will also make available interim releases of newly integrated software capabilities to both customers and the X2000 First Delivery Project for evaluation and test.

Use of OOA/OOD technicques in MDS

The MDS team is implementing MDS software using object oriented analysis and design techniques. The team is using UML (Unified Modeling Language) to communicate their analysis, design, implementation and test decisions. The team is using an OOA/OOD case tool that implements analysis, design, and implementation models using UML notation. The team's OOA/OOD consultant, Bruce Douglass, defines a system model as "an organized, internally-consistent set of abstractions that collaborate to achieve a system description at a desired level of detail and maturity".[3] Throughout an MDS lifecycle the MDS team will develop and/or refine and update various system views (models) of the MDS within their case tool. Each model is another view of the underlying system and is not independent of the other views.

There are several OOA/OOD software case tools available commercially. The MDS team has decided to use a tool called Rhapsody (this tool is produced by I-Logix). The MDS team is currently capturing their analysis and design decisions as a set of models within this case tool. This tool is also capable of generating code from some of the analysis and design models developed within it. The team expects to auto-generate code to do analysis and design model verification. It has not yet been decided whether or not the tool will be used to produce the system's first implementation models although in the interest of supporting BFC missions this is an eventual goal of the MDS team.

The MDS team expects their tool and OOA/OOD methodologies that it supports will help them with system verification and with maintaining consistency between work products as domain teams move through multiple cycles of the MDS inner loop. The Rhapsody case tool enables MDS developers to produce executable models of the underlying system throughout its development. Design concepts can be implemented lightly and executed within the tool as an early check of their feasibility and validity. MDS analysis models (primarily UML sequence diagrams) can also serve as inputs for test verification scenarios once the design has been translated into code. The MDS team expects that the use of the Rhapsody case tool in conjunction with OOA/OOD methodologies will result in a more rigorous and timely mission software development effort than those of past mission projects. The core analysis and design models that the MDS develops within Rhapsody can be combined, instantiated and extended in different ways by various mission customers of MDS to achieve their own unique mission needs. This approach to mission software development should contribute significantly to reduced software lifecycle costs for the new BFC missions at the lab.

MDS Requirements Definition

The MDS team is using UML in their Rhapsody tool to do requirements analysis. Traditionally JPL deep space missions have captured requirements as textual statements. Project requirements are identified via a hierarchical requirement analysis effort that proceeds along hardware lines with project objectives forming the top tier of the decomposition. Mission requirements are

captured and then expanded and decomposed into flight and ground system requirements. Flight and ground requirements are further decomposed into flight and ground subsystems. A flight system might decompose into a spacecraft, its science payload and the launch vehicle. The spacecraft is further broken down into a number of hardware based engineering subsystems such as the command and data handling subsystem, the attitude control subsystem, the telecommunications subsystem, and so forth. Detailed hardware and software requirements are then identified for each of the subsystems. In the past requirements were captured manually in various hierarchical system and subsystem requirements documents. Very recently mission projects at the lab have opted to use a requirements tool, DOORS, to capture their project requirements. This tool contains a requirements database and supports links between requirements at different levels of a requirements hierarchy. This tool has not changed how mission projects do their requirements decomposition.

The MDS team is capturing MDS system and subsystem requirements in Rhapsody. The MDS requirement analysis approach is a black box functional requirement analysis approach. It captures the capabilities the current MDS development effort needs support in order to achieve typical mission software capabilities for its first mission customers. Rather than capturing requirements as textual statements however MDS engineers capture requirements as UML use cases and their accompanying sequence diagrams. MDS use cases and sequence diagrams collectively capture externally visible functions and behaviors of the MDS. This is customer oriented view of mission software requirements that says nothing about how MDS software will be designed to achieve these capabilities. Since MDS customer missions are capturing their mission requirements as textual statements in DOORS the MDS project has been looking into how MDS use cases can be mapped to customer project textual requirements statements in DOORS. The Rhapsody tool currently supports the export of the use case portion of its requirement analysis model into a DOORS database. MDS customers can link exported MDS use cases to the appropriate textual statements in their project DOORS database to show requirements tracing between MDS and the mission project. A future revision of the Rhapsody tool is expected to also export sequence diagrams into a DOORS database. This will allow a more complete mapping of requirements statements between the two tools than is currently supported.

Top level MDS use cases do not always distinguish between hardware and software capabilities. In some cases a use case denotes a capability that must be jointly implemented by hardware and software. The first MDS mission implementation will be on avionics hardware supplied by the X2000 1st Delivery Project. MDS and Europa Orbiter, the first MDS mission customer that will use X2000 1st Delivery Project avionics hardwar, are working with X2000 1st Delivery Project avionics engineers to make the appropriate hardware software trades for functions that will be implemented in both hardware and software.

Lower level MDS use cases will map to MDS software subsystems. During detailed software development (inner loop iterations) MDS domain teams will develop use cases and sequence diagrams to capture the black box functionality of their software subsystems. This analysis will occur after a number of system level design decisions have occurred and so will only be black box relative to the subsystem to for which they are being developed.

Lifecycle support for Early Hardware and Delayed Software Decisions

Typically deep space missions require early commitment to flight hardware in order that this hardware be designed, built (either built in-house or procured), integrated, and tested in time for launch. This is still true for current JPL missions. The time that current BFC mission projects have to make hardware requirements, design and build decisions is significantly shorter than that past mission development efforts such as Galileo and Cassini. The MDS incremental systems engineering approach accommodates the need for mission projects to make hardware decisions early in their development lifecycle. The MDS lifecycle approach supports the development and finalization of top-level black box system and subsystem functional requirements within the elaboration segment of the MDS outer loop cycle.

Once top level functionality is determined and hardware software partitioning agreed to as part of a systems engineering activity in the MDS outer loop cycle the MDS software development effort can proceed iteratively. The inner loop lifecycle allows software designers and implementers to add software functionality in stages. Developers can stage the addition of capabilities according to the dictates of their particular mission. Typically missions require early development of spacecraft flight software that will provide launch and fault tolerant cruise control during the earliest part of the mission. Many missions however can delay development of their mission science and/or technology software until just prior to the arrival of the spacecraft at its destination. There may be years of cruise time for the development this software. Both the MDS lifecycle and the MDS component based software architecture

will facilitate a long duration, iterative software development effort for missions that need this approach.

Mission Software Implemented as Single Software System

In the past both JPL line management and JPL deep space mission projects have contributed to rather fragmented mission software development efforts. Both organizations have tended to view software as internal to the flight or ground hardware subsystem in which it resided and a subset of that subsystem's capabilities. This prevented cross-cutting and/or common software capabilities from being dealt with systematically. It also contributed to a duplication of software development effort.

In previous mission software development efforts JPL standards for software development processes were sometimes unevenly applied to the development of software under the control of a hardware focused subsystem development effort. This resulted in software products that varied widely from subsystem to subsystem in quality, timeliness, maintainability and operability. Additionally, while each subsystem development effort involved the implementation of some software functions unique to the subsystem, it usually also involved the implementation of a large set of software functions that were common to many software subsystems in the project. However because each subsystem was implemented by a separate organization the software teams for each tended to implement the common software functions differently. Only the external software interfaces between subsystems were standardized. And usually JPL systems engineers spent a great deal of time and effort designing, documenting and verifying the implementation of these interfaces with the participating subsystems. This approach to mission software development at the lab resulted in unnecessarily large software development, maintenance and operations costs for a number of past JPL missions. In the era of BFC missions it is an approach that the lab can no longer afford.

The MDS team has chosen to develop all mission software as a single mission software system. This approach allows common but distributed mission software capabilities to be identified and handled globally. Common software functions that would have been duplicated in many of the mission subsystems of past missions can now be identified and assigned to a single domain team to develop. The team produces one set of source code that is then instantiated in multiple software application subsystems within the MDS. Unique software functions will also be identified and assigned to an MDS domain team with expertise in this functionality (some examples are spacecraft attitude control, navigation, and telecommunications). Whether the software is common or unique all MDS development teams will follow the same software development processes. Developing mission software as a single system within one organization should enable the production of cost effective, quality mission software products that avoid the software inconsistencies and duplications of previous mission software development efforts

Executable Models

The MDS case tool, Rhapsody, includes the capability to generate code from its models. One of the goals of the MDS team is to be able to auto-generate real mission software from the models developed in this tool. This is one of the ways in which software lifecycle costs can be significantly reduced for lab based BFC missions. The MDS team is expecting to also use the auto-code generation feature of Rhapsody to generate proto-type code to check the correctness of their evolving models. For instance the team can generate proto-type code to examine the feasibility of a particular design option. The team also can use Rhapsody analysis models (use cases and their accompanying sequence diagrams) as test inputs to system design models to verify the correctness of the later. In short, by using Rhapsody code generation capabilities the team can cross-check their various model views of the evolving mission software system throughout the development of this system. This approach should save development costs by catching analysis, design and implementation flaws and inconsistencies earlier in the project lifecycle than they might otherwise be caught.

Several JPL mission projects are using another I-Logix tool, Statemate, to perform system analysis and design trades. The Outer Planets / Solar Probe Project is one of them. The MDS team has discussed the possible integration of the Statemate and Rhapsody tools with I-Logix representatives. One goal of this effort would be the capability to transfer system model analysis data from Statemate into Rhapsody automatically. This would help to maintain consistency between system analysis models generated in the Statemate tool with software design and implementation models generated in the Rhapsody tool. Another possible result of this merger would be the capability to exercise Rhapsody and Statemate models against one another to do analysis and design trades in a simulated system environment before any major hardware commitments were made. An integrated model based development environment has a lot of potential for reducing mission lifecycle costs. It will be interesting to see what the I-Logix team is able to achieve if and when they integrate these two tools.

MDS support for Reuse and Adaptation

The analysis and design models that the MDS team develops in Rhapsody will be central to the MDS team's reuse strategy. MDS customers will use existing MDS analysis and design models to form the base from which they will build up their own mission unique software system. Since the delivered MDS system will be a collection of completed and partially completed generic mission software capabilities MDS customers will have to adapt and extend portions of the original system to make it fit the needs of their particular mission. MDS plans to facilitate customer adaptations by providing a series of adapter guideline documents. Additionally the MDS team is providing customer missions with an adaptation example in the form of the MDS reference mission.

The MDS team needs to develop a mission software system that can be adapted to a variety of different hardware platforms. The avionics hardware platform that is being supplied by the X2000 1st Delivery Project to MDS mission customers contains processors that use different versions of COTS operating system software and different C++ compilers. MDS customers want to be able to re-host MDS applications. They want to be able to move specific MDS software capabilities to memories and processors other than those chosen by the MDS team for the MDS reference mission. By implementing a real-time extension of the Common Object Request Broker Architecture (CORBA) the MDS team will provide mission customers with capability to re-host MDS developed capabilities. The ORB hides messaging details between applications and between communicating components within a specific application. Applications using the ORB for messaging need not know the location of the software with which they are communicating. The ORB software will allow MDS customers to instantiate MDS applications in locations and in combinations other than those chosen for them by MDS team for the MDS reference mission.

MDS CORBA software should also facilitate post launch migration of MDS based software functionality if and when an MDS mission customer decides to migrate a ground based capability to the spacecraft. The MDS team may also make use of CORBA capabilities to develop spacecraft fault protection and recovery algorithms that can dynamically re-map critical software functions to other hardware elements in the event of a failure of a critical hardware element.

The MDS team will also provide low level hardware proxy software to interface with underlying hardware. This approach should hide hardware software interface details from higher level application code. Application code that needs hardware information will interface to the hardware proxy software. This code will hide all the hardware interface details from the application code. MDS proxy frameworks should be easily modified and/or replaced by mission customers when their hardware differs from that assumed by MDS developers for the MDS reference mission. For example the MDS team plans to include a camera in the MDS reference spacecraft. MDS developers will base their proxy interface code for this camera on camera hardware used in a recent JPL mission. The proxy code and the camera hardware it interfaces with will probably not be the same as that used by MDS mission customers. However mission adaptation of MDS camera proxy code to interface with a real mission camera should be relatively straight forward and should not impact the application code sitting above it.

Summary

The MDS project is not very far into its first circuit of its outer loop cycle. It is much too early in the MDS development effort to predict the success or failure of the MDS approach for future mission software development at the laboratory. This paper has described MDS software lifecycle processes and how these processes should contribute to the success both of the MDS project and of MDS mission customer projects. The MDS team's progress will have to be monitored and re-evaluated on a frequent basis as the team proceeds through their lifecycle for the first time. Some adjustments to the processes and tools may need to be made as the team moves forward.

The MDS development approach is very ambitious. It pilots changes to mission software development that are not only technical but political. MDS customer project personnel and their managers are not familiar with MDS software development processes, methods or tools. Nor are these projects organized to accommodate a unified mission software development effort. The MDS project will not be successful if it is only successful in producing an MDS reference mission system. The MDS team will need to work closely with initial MDS customer mission projects in order to assure that these customer projects successfully adapt MDS to their particular mission needs. The MDS approach to mission software development has a lot of potential for improving the way mission software is developed across the laboratory. MDS needs successful customers to realize this potential.

Acknowledgments

The work described in this paper is being carried out at the Jet Propulsion Laboratory, California Institute of Technology as part of process engineering for the Mission Data System Project.

REFERENCES

1 D. Dvorak, A. Sacks, R. Rasmussen, "Software Architectural Themes in the Mission Data System," AIAA conference paper, AIAA Space Technology Conference, September 1999

2 B. P. Douglass, Real-Time UML: Developing Efficient Objects for Embedded Systems, Addision-Wesley, 1998

3 B. P. Douglass, Doing Hardtime: Developing Real-Time Systems with UML Objects, Frameworks and Patterns, Addision-Wesley, 1999