Reusable, Extendible F light Software for a Planetary Spacecraft' Prototype Testbed

Sanford M. Krasner*
Jet Propulsion laboratory
Pasadena, CA

Abstract

As part of the "Faster, Better, Cheaper" paradigm for NASA missions, the Jet Propulsion Laboratory is developing a Flight System Testbed for prototyping and early integration of future planetary missions, This paper describes the development of a set of reusable, extendible spacecraft flight software to be used as a basis for prototypes in the testbed. This effort has focused on identification and irnplementation of functions which are common across multiple missions. This effort has also developed an intertask messaging system which supports modification of existing functions, additions of new functions, and porting to various computation and I/O architectures. This paper also identifies a number of other JPL activities which support standardization and reusability of planetary spacecraft designs.

Background - Re-engineering and the Flight System Testbed

NASA has challenged the planetary science community (including the Jet Propulsion Laboratory) to develop "faster, better, cheaper" planetary missions, with spacecraft development and launch costs of less than $150 million, Responses to this challenge include the Discovery mission proposals, and the proposed 'New Millennium' program, which envisions a fleet of small, highly-autonomous spacecraft,

JPL is also engaged in a re-engineering effort, to shift from a serial, hierarchical development process to a concurrent, parallel process. In this process, spacecraft design and early integration are conducted in parallel, along with spacecraft operations development. In addition, this re-engineering process is encouraging the reuse of designs from mission to mission, the use of industry standards, and the use of commercial off-the-shelf (COTS) products,

As part of this re-engineering process, JPL is developing the Flight System Testbed (FST)[1]. The FST will provide representative spacecraft functionality, including "flight-like" computers, software and I/O interfaces, as well as simulations of spacecraft hardware and dynamics, By providing a 'flight-like" environment which is easily tailored to simulate a particular spacecraft, the FST will support early prototyping of system design issues, and early integration of mission equipment. As a multi-mission testbed, the FST will also encourage design reuse, and the application of industry standards. Various activities to support this goal include:

- Development of an easily-tailored spacecraft dynamics simulation engine, and a supporting library of actuator and sensor models[2],
- Study and recommendations for standard avionics interfaces, including bus standards
- Simulations of spacecraft telecommunications, power, propulsion, attitude determination and control, and command and data handling equipment;
- Simulations of science instruments, including data editing, compression, and progressive transmission,[4]
- Integration with a CCSDS-compatible ground control system,
- Development of a reusable, extendible set of flight software, including the development of a reusable set of high-level spacecraft commands. This also supports the development of a standard,

'Member of Technical Staff,
Avionics System Engineering Section
email:skrasner@jpl.nasa.gov

reusable concept of spacecraft software design and implementation, and the development of standard ground operations tools and procedures.

Identification of Common Spacecraft Functions

In order to provide reusability of spacecraft flight software for future missions, a study was made of the software functions implemented on several planetary spacecraft including spacecraft in pre-project studies, under development, or in operation, This list of spacecraft now includes: Galileo, Magellan, Cassini, Mars Observer(MO) and Mars Global Surveyor (MGS), Mars Pathfinder, Pluto Express (formerly Pluto Fast Flyby), and Clementine. These are referred to below as the "referenced missions."

These spacecraft span a wide range of missions, technologies (from circa 1976 to 1994) and development paradigms. Galileo and Cassini are complex, multiple-instrument outer-planet orbiters developed at JPL. Magellan was a Venus-orbiting radar mapping spacecraft, with a very small instrument complement; Magellan inherited a large portion of the Galileo avionics, as well as the Galileo Command and Data Subsystem flight software. Mars Observer was a Mars-mapping mission, with significant inheritance from an Earth-orbiting weather satellite; MGS inherits the MO avionics and most of its flight software, Pathfinder is a single-string, spin-stabilized Mars lander and rover mission with an emphasis on technology demonstration, developed in the 'faster, better, cheaper" mode. (Only the spacecraft/lander software was examined in this study.) Clementine was a lunar mapper and asteroid flyby mission, with a focus on ballistic missile defense sensor demonstration and evaluation, and was also developed in a 'faster, better, cheaper' mode. Galileo, Cassini, and Pathfinder are/were built by JPL. Magellan, Mars Observer and MGS are/were developed by Martin Marietta, as a contractor for JPL. Clementine was developed by Naval Research Laboratory for the Ballistic Missile Defense Organization. Pluto Express is a pre-project study being conducted at JPL.

In spite of the range of mission types, technologies, and development organizations and paradigms represented by these spacecraft, there is much functional similarity among the flight software for these missions. It is feasible to identify and implement a set of functions which are likely to be reusable on future planetary missions. In order to support reusability of flight software and mission operations tools, a corresponding set of common spacecraft commands is also being identified. A high-level picture of the common functions is represented by the dataflow diagram in Figure 1.

The common spacecraft functions study identified a "3-level" classification of functions (see Figure 2). The bottom layer consists of hardware- and mission-specific software components, which are dictated by the specific hardware sensors and actuators used. The reusable software design has attempted to encapsulate the hardware-dependencies of this layer, so that it presents a standard set of interfaces to the layer above. (This may be considered as defining 'classes" of virtual devices, which are customized for particular applications.)

The middle layer is largely independent of specific spacecraft hardware and mission, and consists of the common functions identified below. Figure 2 shows a subset of these functions.

The top layer consists of the high-level coordination of middle-layer spacecraft functions to accomplish mission objectives, including system fault protection responses to coordinate spacecraft resources in the presence of faults. This layer has not been as clearly identified in the referenced missions, since coordination of spacecraft resources has been typically accomplished by ground sequencing. The reusable flight software provides the 'primitive' functions invoked by the high-level coordination. The reusable flight software also provides a control language which may be appropriate for implementing spacecraft coordination.

The common functions are described below:

1.) Uplink Processing

These functions include all processing needed to recognize and process ground commands addressed to the spacecraft, and to distribute commands to the onboard applications which will execute them. Although each of the referenced missions used somewhat customized uplink formats and protocols, future spacecraft are expected to adhere to the uplink standards of the

Consultative Committee for Space Data Systems (CCSDS).

2.) Sequence Execution and Spacecraft Coordination

Due to the long radio communications delays and infrequent ground contacts for planetary missions, each of the referenced missions includes a sequencing or scripting capability, which allows the ground to put command loads onboard for execution at later times. These scripts are implemented in 'sequence languages" of different levels of sophistication. All implement at least a minimum set of capabilities, including the ability to issue commands at absolute *or* relative times, and to execute multiple scripts concurrently. Clementine used the Spacecraft Control Language (SCL) which is available commercially. JPL is currently evaluating SCL and other languages as candidates for a standard spacecraft control language.[6]

Until now, spacecraft activities have been orchestrated by ground commands, by way of the sequence language. As future spacecraft become more autonomous, this "sequencing function" will evolve to a high-level 'spacecraft coordination" function. To achieve this transition, supporting languages and execution engines will be required. Various studies at JPL are currently examining autonomy requirements and designs to support future missions.[7] The FST reusable flight software, although not autonomous in itself, will provide a facility for developing and testing autonomy designs.

3.) Attitude & Trajectory Determination

Although the sensors used vary somewhat, each spacecraft includes hardware and software to determine the spacecraft's orientation with respect to inertial space. Except for Pathfinder, all of the referenced spacecraft use gyros to propagate inertial attitude. The spacecraft use either imaging star trackers (Cassini and Clementine) or slit star scanners to determine spacecraft orientation with respect to the fixed stars, and to correct for gyro drift.

With the availability of imaging cameras, large on-board memories, and powerful processors, and with the increasing need for spacecraft autonomy, JPL and others are investigating the addition of autonomous optical navigation, target tracking[9], and trajectory correction[10]. These functions will fall under the general category of trajectory determination.

4.) Attitude & Trajectory Control

All spacecraft include functions to control the spacecraft attitude and trajectory. These functions may also include logic necessary to control articulations, such as instrument platforms, solar arrays, and antennas. These functions are probably the most hardware and mission-dependent, since they are strongly related to spacecraft characteristics such as: mass properties, thruster locations and types, articulation axes, actuator performance, etc. JPL is developing a set of advanced design processes and tools [11] to support development of attitude and trajectory determination and control designs. One aspect of this program is an autocode generator to implement these designs. The reusable flight software task will identify the architecture and interfaces to accommodate these autocoded components.

This category of functions also includes inertial vector propagation, which is responsible for the propagation of ephemerides for other celestial bodies relative to the spacecraft. This utility function provides inertial directions to the Earth, for uplink and downlink antenna pointing; to the Sun for attitude initialization; to target bodies such as asteroids or planets for observation.

5.) Instrument Control and Data Collection

The *raison d'être* for planetary spacecraft is to return data from science instruments. Each spacecraft provides functions to command science instruments and collect measurements for transmission to the ground. For Pathfinder and Clementine, instrument operation, data collection and formatting is conducted by the same processor which executes spacecraft engineering functions; for the other missions, these functions are handled by computers dedicated to the instruments.

The Flight System Testbed is funding development of a generic instrument simulator. The reusable flight software will provide a messaging 'gateway' to send commands and ancillary data to instruments, and to collect instrument data in telemetry packets.

6.) Telemetry Data Handling

Each of the examined missions provides functions for collecting data from onboard applications, storing it onboard, and transmitting data from storage or in real-time. Although the earlier missions used custom downlink protocols, the CCSDS Telemetry recommendations have established standards for telemetry data structures.

The CCSDS telemetry standards allow individual applications to generate self-identifying packets of telemetry data, and eliminate the need tor centralized time-commutation telemetry generation services. These self-identifying packets also enable the use of intelligent instruments and data compression algorithms which may not produce telemetry at predictable rates and times.

With the improvements in random-access memories, spacecraft designs have moved from digital tape recorders (Galileo, Magellan, Mars Observer) to large RAM or solid-state recorder storage (MGS, Pathfinder, Clementine, and Pluto). Random access to telemetry storage enables a number of features which cannot be accomplished with serial input/output tape recorders. These include the ability to interleave different categories of telemetry on playback, to retrieve telemetry in a different order than it was recorded, and to release pieces of telemetry storage as they are played back, These capabilities are exploited in the Pathfinder design. The reusable flight software will provide flexible telemetry-handling capabilities which take advantage of the packet telemetry standards and random access storage.

The telemetry handling functions are also responsible for generating the downlink data stream from real-time and playback data. Downlink data is typically encoded (e.g. Reed-Solomon coding, convolutional encoding), but encoding is not included in the reusable flight software.

7.)Time Reference

Each of the spacecraft examined here provides a central spacecraft time reference. This reference is used as a basis for absolute-time defined sequencing, and for time-tagging telemetry data generation, The time reference functions also work with the telemetry handling functions to provide time calibration data to the ground system. l-he ground system correlates spacecraft time to telemetry receive times to determine the spacecraft-universal time relation. This correlation is used both in command planning (sequencing) and in telemetry analysis to correlate spacecraft activities to spacecraft position in its trajectory or relative to celestial bodies.

For some missions, the spacecraft time reference functions have also imposed a master time schedule on spacecraft activities. This has been especially true for missions which use a central time-commutated telemetry system. The reusable flight software design provides a central time reference but does not impose a master time schedule. This allows each application to use the time schedule most appropriate for itself.

8.) Internal State Control

Each spacecraft carries functions to measure temperatures, pressures, voltages, currents, and other discrete and analog measurements, The spacecraft also provide commands to turn loads on and off, set discretes, and fire pyrotechnic devices. There have been some limited implementations of closed-loop thermal control, and Clementine uses software to control the propulsion system pressurization. The reusable flight software will provide the ability to read discrete and A/D measurements, and to command power switches.

Fault Detection Isolation and Recovery

Due to the long communication round-trip times, infrequent ground contact passes, and time-critical activities (e.g. orbit insertions, flybys) characteristic of planetary missions, planetary spacecraft typically carry extensive fault detection, isolation, and recovery designs. (This is not true for the “faster, better, cheaper” Clementine and Pathfinder spacecraft.) The initial design of the reusable flight software does not include provisions for fault protection logic, as it is known at JPL. This decision was taken in order to focus on nominal mission capabilities, and to focus on mission-independent spacecraft aspects, JPL is conducting a study to define a reusable fault protection architecture. It is recognized that fault protection designs will also have to be made reusable to reduce their high development cost, The reusable flight software

design will be revised to reflect the results of this study.

Message-Oriented Architecture

To enhance reusability, the reusable flight software is designed as a set of concurrent tasks., which communicate primarily by sending messages to each other. For repetitively scheduled tasks, such as estimators and control laws, a timer service is provided to generate periodic 'wake-up" messages, which are handled through the standard messaging functions.

The decision to use messaging extensively was driven by several factors:

.1) The need to define clean and explicit interfaces between software components to enhance reusability. This eliminates the use of a shared memory interface.

.2) The need to allow tasks to run concurrently at different priorities while allowing for communication between them.

.3) The need to encapsulate hardware-specific operations in device drivers which provide reusable interfaces to other software components. For instance, interrupt handlers and hardware-specific conversion tasks are used to encapsulate the spacecraft-specific aspects. Hardware-independent messages are then presented to the rest of the flight software. (For example, a gyro data handler would acquire gyro data, scale to engineering units, and convert to spacecraft body coordinates, The resulting gyro data message would be hardware-independent. An uplink interrupt handler would be responsible for acquiring command codeblocks from the uplink hardware, and would present CCSDS-compatible codeblocks to the reusable uplink software.)

.4) The need to add new generators and consumers of data without modifying existing software components. This is required to support prototyping of new functions with minimal impact on existing software.

,5) An anticipated need to support flight software distributed across multiple processors.

.6) "The availability of commercial real-time operating systems which support concurrent tasks and intertask messaging, In particular, Wind River Systems' VxWorks[12] has been chosen as the standard real-time operating system for f ST real-time target development.

.7) The prevalence of intertask messaging in the referenced spacecraft designs (Clementine, MO/MGS, Pathfinder).

The reusable flight software provides the basic structure for prototyping of new software functions to support new missions. To simplify the integration of new software, a message-handling layer was developed to allow tasks to broadcast messages on the equivalent of a "software bus". The task generating the message does not need to direct the message to particular receiving tasks. Tasks which need to receive a particular type of message may subscribe to that "message subject". The message handling layer is responsible for knowing which tasks have subscribed to a particular type of message, and for routing the message to all receiving tasks.

This message handling layer is known as the Task Remote Asynchronous Message Exchange Layer (TRAMEL)[13]. TRAMEL is written in C for portability to other machines, and supports all intertask messaging functions supplied by VxWorks, including pipes, sockets, and TCP/IP interprocessor communications. TRAMEL is similar to the Standard Asynchronous Message (SAM) system used on Clementine[14], with the added capability of multiple tasks receiving the same message.

Initial Implementation and Future Plans

The FST simulation environment consists of a network of Sun workstations with simulations of spacecraft hardware and dynamics running on different workstations. The first target system for the reusable flight software will be a Heurikon 68040 board. (The software is designed in a machine-independent fashion, so choice of the initial target should not affect its portability to other processors.) The software will be written in C, using VxWorks as the real-time operating system. The interface between the 'flight' processor and hardware dynamics simulation is an Ethernet connection. The VxWorks socket utilities will be used to provide the hardware interface portions of the device drivers.

SCL will be integrated to provide the spacecraft sequencing and coordination function in first quarter of 1995. This requires porting the software to an R3000 processor . Later in 1995, a 1553B

interface will be installed as the processor interface to simulated hardware in place of the Ethernet connection.

Summary

JPL is re-engineering its process of planetary spacecraft development. One part of this re-engineering is the development of flight software which supports early prototyping and reusability of flight software and ground operations from mission to mission. This software is currently being implemented and demonstrated in a Flight System Testbed for prototyping support of future missions.

The work described in the 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]Casani, E. Kane and Nicholas W. Thomas, 'The Flight System Testbed", Space Instrumentation and Dual-Use Technologies, SPIE - The International Society for Optical Engineering, Orlando, CA, July 4-6, 1994

[2]Jain, A. and G. Man, "Real-Time Simulation of the Cassini Spacecraft Using DARTS: Functional Capabilities and the Spatial Algebra Algorithm," in 5th Annual Conference on Aerospace Computational Control, Aug. 1992.

[3]Caldwell, Douglas, and Savio Chau, "Spacecraft Information Systems: Principles and Practice", JPL. Internal Report, July 7, 1994

[4]Lee, Meemong, Alan Mazer, Andy [loden, Steve Groom, "Instrument Simulation Testbed', JPL Internal Presentation, Dec. 7, 1 993

[5]Krasner, Sanford M., "Preliminary Spacecraft Information System Guidebook",JPLIOM3132-93-428, Nov. 30, 1993

[6]Rohr, John A., et. al., "Spacecraft Command Language (SCL) Study Report', JPL report D-12206, Oct. 26, 1994

[7]Ahmed, A., Aljabri A. S., 'Demonstration of On-board Maneuver Planning Using Autonomous SAN Architecture", Annual AIAA/USU Conference on Small Satellites, August 1994.

[8]Vaughan, Robin M., "Pluto Fast Flyby Autonomous Optical Navigation System Functional Description" JPL IOM 314.8-905 July 20, 1994

[9]C.-C. Chu, D. Q. Zhu, S. Udomkesmalee, and M. 1. Pomerantz, "Realization of autonomous image-based spacecraft pointing systems: planetary flyby example." In Acquisition, Tracking, and Pointing VIII (M. K. Masten, L. A. Stockum, M. M. Birnbaum, and G. E. Sevaston, Editors), Proc. SPIE 2221, pp. 27-40.

[10]"System Algorithms for Autonomous Navigation: Application to the Rendezvous Phase", JPL IOM 314,3-1120, Sept. 22, 1994

[11]Man, Guy K., "Advanced Design Processes and Tools (ADEPT)", Internal Presentation, Oct. 25, 1994

[12] *Vx Work Programmer's Guide,* Wind River Systems, Inc., Alameda, CA, 1993

[13]Scott Burleigh, "ROME: Distributing C++ Object Systems", IEEE Parallel and Distributed Technology, Vol. 1, No. 2, May 1993, pp. 21-32.

[14]Wildermann, C., et, al. 'Flight Software", Clementine Engineering & Technology Workshop, lake Tat ıoe, NV, July 18-19, 1994

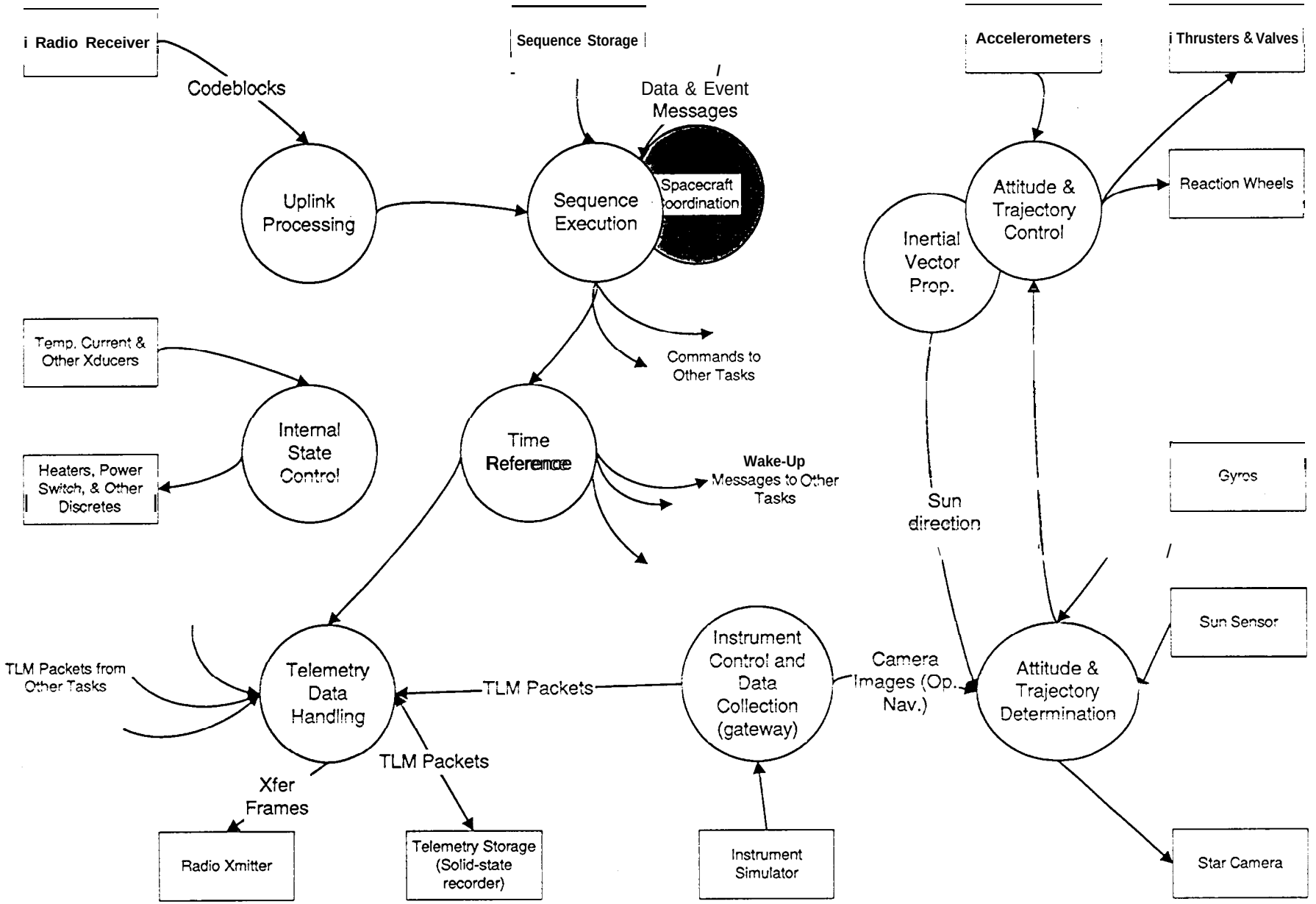

Figure 1
Common Spacecraft
Functions

Figure 2-
Common and Mission-Specific Functions