Chapter 4

THE SPHERES LABORATORY FOR DSS RESEARCH

New scientific and economic objectives are creating a demand for small satellites capable of autonomous formation flight. Current missions, such as separated spacecraft interferometry, require large diameter synthesized apertures than today's monolithic satellites can provide since they are limited in size by launch and deployment capabilities. A group of smaller satellites flying in an array would provide the desired improvement; furthermore, the smaller size of individual satellites and the increased modularity of a constellation system would result in a reduction of launch and maintenance costs.

SPHERES is designed to create a testbed to demonstrate the viability of autonomous formation flight control algorithms. SPHERES provides a facility with six degrees of freedom to evaluate the dynamics of a multiple satellite system. It also tests the ability of a constellation of independent objects in a microgravity environment to interactively communicate, maintain position, run diagnostics, regroup after disturbances, and move to commanded locations.

SPHERES was designed specifically for operations in the ISS, following the guidelines of the MIT SSL Laboratory Design Philosophy presented in Chapter 3. Because of direct involvement and deep knowledge of SPHERES, it is a primary candidate to demonstrate how to best implement the features of the philosophy and ensure the best use of the ISS.

This chapter first presents the science requirements of SPHERES as well as the different constraints imposed by operations aboard the ISS. A introductory description of the design of the facility and its sub-systems follows. Next, the chapter discusses how the different sub-systems of SPHERES implement the features called for in the design philosophy.

4.1 SPHERES Problem Statement

The primary goal for SPHERES is to create a testbed for the development of formation flight and docking algorithms for separated spacecraft systems. The requirements were based on both the primary goal and the MIT SSL Laboratory Design Philosophy. The first step in the design process was to develop a clear set of requirements for the facility, and understand the constraints of the operational environments.

4.1.1 SPHERES Requirements

The initial design of SPHERES had its first milestone in the Spring of 1999 when the undergraduate senior class presented a preliminary design. The science requirements realized from the initial problem conception are [SPHERES, 1999]:

1. Develop a set of multiple distinct spacecraft that interact to maintain commanded position, orientation, and direction.
2. Allow reconfigurable control algorithms, data acquisition and analysis, acquisition of a truth measure.
3. Enable the testbed to perform array capture, static array maintenance under disturbances (attitude control and station keeping), and retargeting maneuvers.
4. Enable testing of autonomy tasks, including fault-detection and recovery, health and status reporting, and on-board replanning.
5. Ensure traceability to flight systems via communication, propulsion, structural, avionics, guidance, control, and power capabilities.
6. Design for operation in the KC-135, shuttle mid-deck, and ISS.

These requirements reflect both the mission objective and the guidelines presented in the MIT SSL Laboratory Design Philosophy. The first, third, and fourth requirement directly

relate to the mission objective of creating a formation flight development facility. The second, fifth, and sixth requirements evolved from the philosophy so that the resulting facility enables technology maturation. Requirements three and four define individual areas of formation flight technology, providing initial insight into the different tasks which are required to demonstrate maturation.

4.1.2 ISS Constraints

SPHERES had to meet constraints for operation aboard NASA facilities. The initial design was considered for flight in the Shuttle Middeck, while the final flight configuration was designed for operations about the ISS "Unity" node and/or US "Destiny" laboratory space. This section presents the main constraints imposed by operation of SPHERES within the ISS, where it was certified for operations.

1. **Crew availability.** While the ISS presents the only space where humans can interact with μg experiments over an extended period of time, crew availability is limited. Even when three humans manned the ISS, SPHERES has been allocated only one US astronaut for operations - therefore SPHERES has to be operational with the supervision of one human.
2. **Safety requirements.** The ISS safety panel imposed strict margins on the safety of the SPHERES satellites and support hardware. These included:
   - Structural - the structure had to withstand a specified impact and prevent shatter; all edges had a minimum radius requirement
   - Compressed gases - any compressed gases needed safety factors from 1.5x up to 3x the operational pressures; triple hardware redundancy was required of all pressurized elements (i.e., two serial hardware failures could not pose a danger to the astronauts or the ISS)
   - Power systems - all electrical power systems require double redundancy from causing harm to the astronauts or the ISS; the major concern in power systems is the start of fires due to uncontrolled currents
   - EMI - electromagnetic emissions were closely monitored; this included a limited range of RF frequencies available for free use
   - Software - any safety-critical software had to be NASA certified
3. **Volume**. The volume of all SPHERES hardware was originally limited to fit within one Middeck Locker Equivalent (MLE). The requirement evolved over time. The need for each individual unit to fit within an MLE remained a

hard constraint; the need for all the hardware to fit within one MLE became a soft constraint.

4. **Mass**. The mass of the SPHERES hardware was also constrained to that allowed in one MLE. As with any space project, the total system mass should be minimized to increase the launch possibilities and reduce cost.
5. **Operations.** The ISS requires remote operations of the satellites, with no direct communications or control from ground. Further, the majority of the test sessions will be conducted independently by the astronauts; real-time communications with the astronauts is only expected in the first two test sessions.

4.2 SPHERES Design Introduction

The SPHERES laboratory for distributed satellite systems consists of five nano-satellites (Figure 4.1), metrology and communications hardware, a researcher interface, an astronaut interface, and a guest scientist program to allow multiple researchers to use the facility. In its final configuration, three of the satellites will be aboard the ISS, where the astronauts will conduct tests in 6DOF. Two units will remain in the ground facilities of the SSL where MIT researchers will tests algorithms prior to up-link to the ISS. The guest scientist program provides a simulation which allows researchers outside of the MIT SSL to develop their initial algorithms in house.

Figure 4.1 The five flight-qualified SPHERES nano-satellites

SPHERES was designed specifically for operation in the shirt-sleeve environment of the MIT SSL laboratory (3 DOF), NASA's KC-135 reduced gravity airplane (short duration 6 DOF), and the International Space Station (long duration 6 DOF). The KC-135 and ISS environments provide 3-D environments to test algorithms that may be directly applied to real satellites. The additional laboratory environment at the MIT SSL enables 2-D experiments to be performed before testing on the KC-135 or ISS, thereby reducing the cost and risk to develop and verify algorithms in the ISS. Figure 4.2 shows an operational concept for SPHERES: the scientist first develops their algorithm using the simulation; the algorithms are then sent to the MIT SSL where tests are conducted with flight hardware in 2D; once tested, the algorithms are sent to the ISS for 6DOF tests.

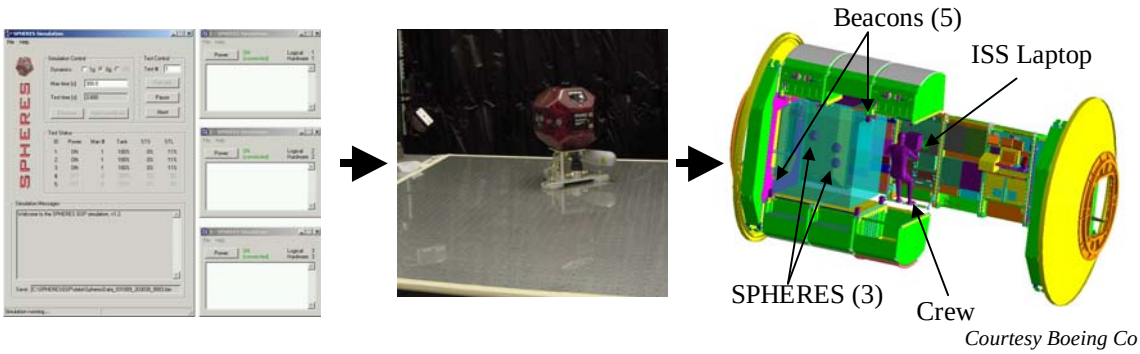


Figure 4.2 SPHERES operational concept

Operating SPHERES in one of the mentioned operational environments requires the following hardware components:

- one to five satellites
- five metrology transmitters
- a communications transceiver
- multiple battery packs
- multiple gas tanks
- a computer with a SPHERES graphical user interface

These hardware elements comprise the package of components which will be delivered to the ISS (a standard NASA supplied laptop is used aboard the ISS, only the software is delivered). The ISS rendering of Figure 4.2 shows this setup graphically. The three satellites, which use the battery packs and gas tanks, will operate inside the blue cube region. The five metrology transmitters, placed on the corners of this region, define the 3D frame of reference. The communications transceiver attaches to the computer via a serial port to store telemetry data, uplink programs to the satellites, and send commands to control tests.

During the design phases of SPHERES the team had to determine a set of requirements that ensure future algorithms will run in the testbed and provide significant results. These requirements include the precision, accuracy, and operational ranges of sensors and actuators, processing power, operational lifetime, and communications bandwidth. Table 4.1 summarizes the resulting quantitative "straw-man" requirements.

TABLE 4.1 SPHERES quantitative operational requirements

Item	Requirement
Translation (1m start to stop)	5s
Rotation (360° start to stop)	5s
Translation accuracy	0.5cm
Rotation accuracy	2.5°
Propulsion lifetime	20s
Power lifetime	90min
Mass (all units + consumables)	24.5 kg
Processing Power	23 MFLOPS
Communications Data Rate	40kbps

To produce results traceable to proposed formation flight missions the individual self-contained satellites have the ability to maneuver in six degrees of freedom, to communicate with each other (satellite to satellite: STS) and with the laptop control station (satellite to laptop: STL), and to identify their position with respect to each other and to the experiment reference frame via a custom metrology system. The laptop control station is used to

collect and store data as well as to upload control algorithms to the satellites. Figure 4.3 shows a picture of an assembled SPHERES unit. Physical properties of the satellites are listed in Table 4.2.

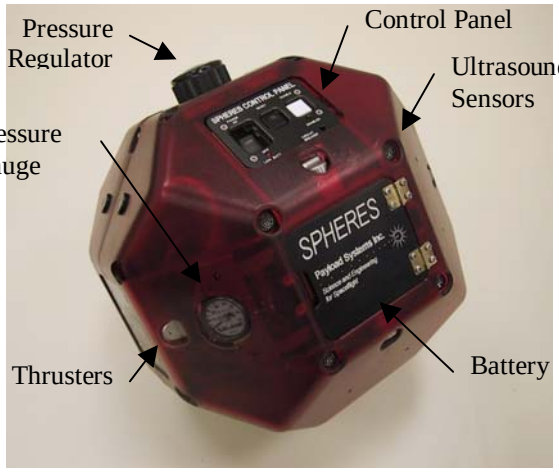


Figure 4.3 SPHERES satellite

TABLE 4.2 SPHERES satellite properties

Diameter	0.25 m
Mass (w/tank & batteries)	4.0 kg
Max linear acceleration	0.17 m/s^2
Max angular acceleration	3.5 rad/s^2
Power consumption	15 W
Battery lifetime	2 h

4.2.1 SPHERES Sub-systems

The SPHERES project was subdivided into six major sub-systems: avionics, software, communications, propulsion, structures, and operations. The avionics sub-system is further divided into processing and support avionics, power, and metrology. The sub-system teams concentrated mostly on the design of the individual satellites, although some sub-systems directly affected the full facility. The avionics sub-system mostly affects the satellites, although the metrology team had to create other hardware. The software team con-

centrated on the operating system that controls the satellites. The communications team had to work both on the hardware within the satellites as well as communication protocols and hardware for the laptop. The Interface/Operations sub-system dealt directly with the whole system, rather than with the satellites. The propulsion and structures sub-systems were limited to the design of the satellites. The sub-system division is summarized in Table 4.3.

TABLE 4.3 SPHERES sub-systems

Sub-System	Scope
Avionics	
Data Processing	Satellites
Power	Satellites
Metrology	System
Communications	System
Software	Satellites
Interface/Operations	System
Propulsion	Satellites
Structures	Satellites

The design of the sub-systems considered the need to satisfy the science goal for formation flight as well as the creation of a broader laboratory for separated spacecraft algorithms. Table 4.3 shows the cases where each of these two goals heavily affected the design of the sub-system. The sections below present short descriptions of the major subsystems and illustrate how each of them fulfills one or both of the goals.

4.2.1.1 Avionics

Figure 4.4 shows an overview of the SPHERES avionics. The figure shows the further sub-divisions of this sub-system. The data processing is the central element. It is surrounded by metrology, communications, propulsion, and power sections which support the other sub-systems of the SPHERES satellite. The metrology system FPGA is an essential element of the avionics system, as it also provides the interfaces for the control panel

and the propulsion system. An external watchdog was implemented to ensure that, if the avionics stop responding, the system is reset. The major elements of the avionics system are described next.

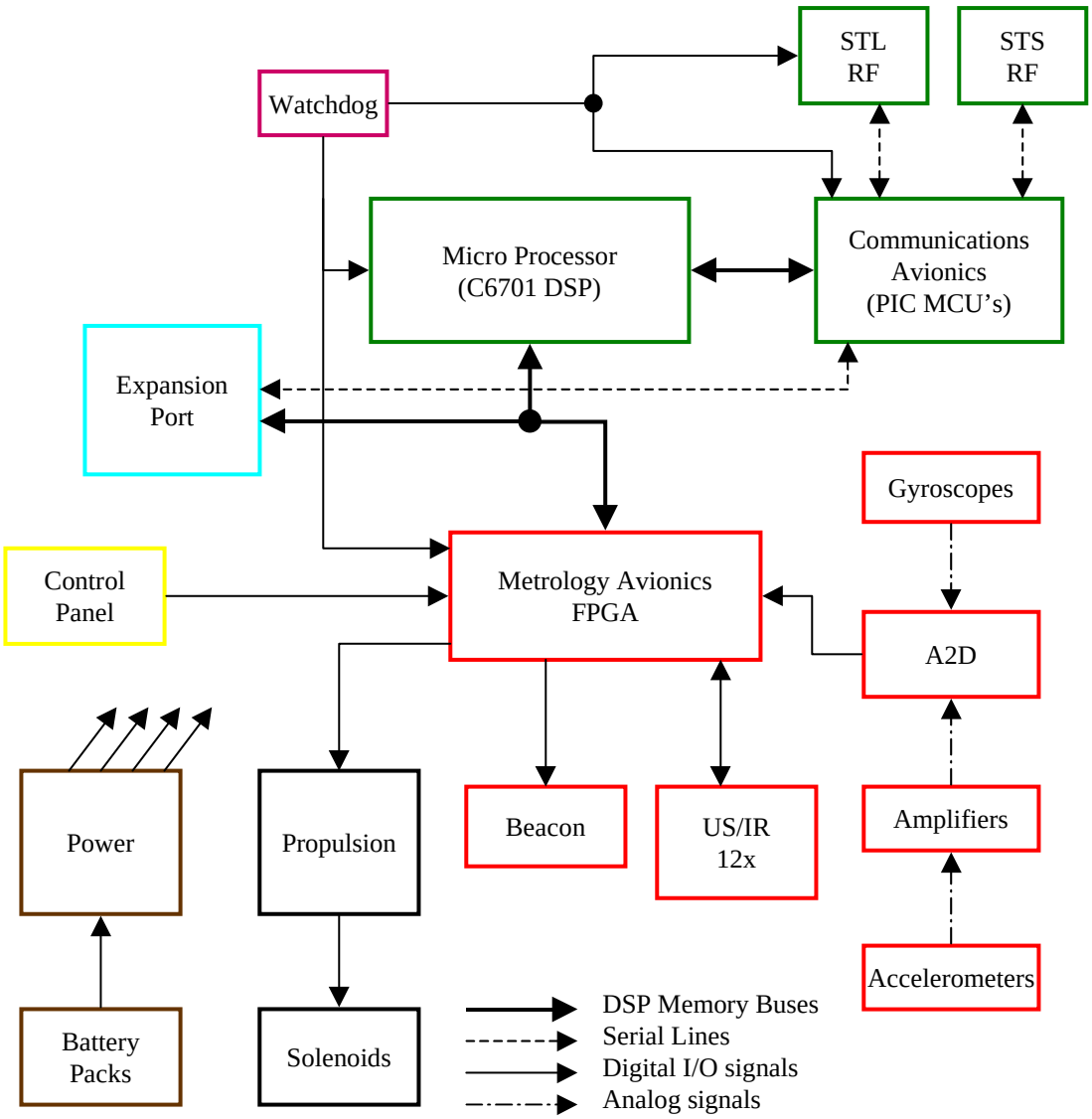


Figure 4.4 SPHERES avionics overview

Data Processing. A Texas Instruments C6701 Digital Signal Processor (DSP) ([TI, SPRS067E], [TI, SPRU189F]) provides the computational power. DSP processors provide multiple features that ensure real-time operation. Further, the DSP processors include

all support functions of a standard processor, allowing it to control the whole unit. The ability of the C6701 to provide between 167MFLOPS up to 1.0GFLOPS, provides significant processing power to prevent being the limiting factor in the performance of the system. The processor is supported with 16MB of RAM and 512KB of FLASH memory. The FLASH memory stores the programs in each satellite. A customized bootloader program allows the FLASH to be reprogrammed during normal operations, allowing unlimited changes to the software.

The original requirement for 23 MFLOPS, based on the estimated needs for formation flight control algorithms, is easily met by the C6701. The selection was ultimately based on the need to provide enough processing power for a wide range of scientists while being compact, low power, and passively cooled. The level of processing power takes into account that some of the algorithms that SPHERES could test have not even been conceived yet.

The metrology sub-system, described below, utilizes inertial sensors (accelerometer and gyroscopes) and a global system (which uses infrared and ultrasound pulses to measure time of flight) to determine the state of the satellite both inertially and with respect to a fixed frame of reference. To perform these calculates the metrology system must support several multiple level interrupts asynchronously with the rest of the software system, which would otherwise consume too much DSP processing time. Further, the system needs analog input lines not available in the DSP. A VIRTEX FPGA [Xilinx, DS001-1] supports the metrology functions. The FPGA handles the asynchronous interrupts of the metrology sub-system and the data capture. The DSP takes the raw information from the FPGA and runs the estimation algorithms.

The rest of the avionics subsystem consists of a propulsion solenoid driver board, a power distribution board, a digital communications board, two RF communications circuits, and the metrology infrared/ultrasonic receiver boards.

Power. The power system for ISS operations consists of two packs of eight AA alkaline batteries per satellite. The packs provide each unit with approximately two hours of operation; once a pack is consumed, it can be easily replaced. The power sub-system provides electrical power to the other subsystems via electronics compatible with the KC-135 and ISS. The power is regulated to provide the necessary voltages for all the subsystems. The total power requirement for a SPHERES free flyer is approximately 13 W. The demonstrated lifetime of the batteries, during operation in both a one-g laboratory environment and the KC-135, is approximately 120 minutes.

For ground-based operations, such as the MIT SSL and the KC-135, rechargeable NiMH battery packs were built. These packs also provide approximately two hours of operations. Due to safety concerns during recharging the rechargeable packs cannot be used in the ISS.

The design of the power sub-system was guided by the requirements for operations aboard the ISS. The goal was to ensure that the primary mission could be accomplished in the ISS. At the same time, the need for replaceable consumables (and rechargeable batteries in ground-based environments) was driven by the MIT SSL Laboratory Design Philosophy.

Metrology. The metrology systems generates real-time estimates of the satellite's state. The metrology measurement system include a global metrology system used to estimate the satellite state with respect to the external reference frame, and an inertial measurement system with accelerometers and rate gyroscopes that is used to measure high-frequency body frame accelerations and angular rates. The global metrology system measures time-of-flight using a combination of infrared and ultrasound signals. The time-of-flight is used to determine the distances between 24 sensors located on the surface of each satellite and five ultrasonic beacons placed at known locations in the work volume. The SPHERES team provides an Extended Kalman filter that uses the inertial and global systems to determine the states of each satellite with respect to the reference frame, although scientists may develop their own estimation routines. Relative state information (e.g. range and

bearing), can be obtained by exchanging the global state information, or by using beacons located on the satellites themselves.

While the metrology system provides essential information for a wide range of distributed satellite systems (DSS) algorithms, the design and implementation were a result of the need to support formation fight.

4.2.1.2 Communications

Each SPHERES unit uses two separate frequency communications channels with an effective data rate of approximately 45kbps per channel. One channel is used for satellite-to-satellite (STS) communications; the other channel enables satellite-to-laptop (STL) communications. Both channels are bidirectional; however, the communication hardware is half-duplex, meaning that only one unit can transmit at a time. The choice of two communications channels closely models the expected operations of future formation flying missions where a high-bandwidth, low-power (short distance) communications link sends data between the units while in space and a separate high-power ground communications link is provided. The two channels of SPHERES are identical in functionality, other than their different frequencies, and therefore the scientists can decide how to best use the two separate channels.

Access to the STS and STL communications channels is controlled by a Time Division Multiple Access (TDMA) scheme. A fixed period of 200ms is shared between the satellites and the laptop (for STL, STS does not include the laptop); the allocated transmission time for each satellite can be configured manually or automatically. The communications module manages transmission and reception of the messages generated by both the SPHERES Core software and the experiment code, such as custom telemetry or command data. If a data transfer is too long for a single packet (32 data bytes), the communications module segments the transmission and sends one packet at a time. The communications module on the receiving SPHERE automatically reassembles the original message from the constituent packets.

4.2.1.3 Software

The software sub-system for the SPHERES satellites is built on a four layer structure, illustrated in Figure 4.5. The software creates an interface for the scientists such that they are never required to program the hardware directly (although it is possible), but rather use higher level functions, simplifying the implementation of their code. The lowest level is comprised of the actual hardware being controlled (e.g., thrusters, RF boards, etc.). The Texas Instruments DSP/BIOS real-time operating system ([TI, SPRU403E], [TI, SPRU423B]), designed for DSPs such as the C6701, is used as the base operating system on the SPHERES satellites. DSP/BIOS provides multi-processing capability, inter-process communication, and a number of input/output management tools. It is the layer which interacts directly with the hardware and manages many of the details for thread and interrupt handling.

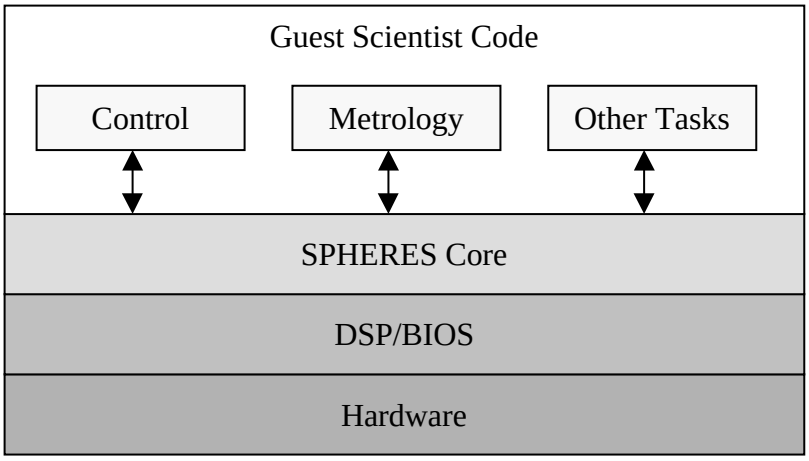


Figure 4.5 SPHERES software layers

The SPHERES Core interface implements multiple distinct execution threads which define the SPHERES Operational environment. This layer implements the basic housekeeping functions required to operate the satellite. These functions run separately from another set of threads designed to execute the test specific algorithms. This separation ensures that activities such as communications and telemetry processing are not affected by any computation-intensive algorithms supplied by the guest scientist. The primary ele-

ments of the SPHERES Core layer are: interfaces to the metrology and propulsion hardware, management of tests and the control interrupt, and management of the communications. The metrology functions capture the data and make it available to the scientist and other SPHERES Core functions, but do not process the data. The propulsion interrupt serves to ensure the thrusters operate as required by the other avionics elements and provides a third level of safety beyond the required NASA safety requirements. The test management functions allow scientists to program multiple individual tests in one program; the functions run the initialization routines and ensure that the tests start synchronously among multiple satellites. The control interrupt management allows the scientist to specify multiple rates without compromising the performance of other threads. The communications core functions implement the communications protocol and ensure compliance with NASA requirements.

The highest level is the actual program implemented by the scientist. To implement their program the software provides six different insertion points for code: two separate periodic, high priority interrupts to collect the metrology IMU and global data; a function to initialize a test, automatically run by the test management part of SPHERES core; a periodic interrupt to run control algorithms; a general purpose background task to be used for functions that require long processing time, but which need not be periodic; and a background task directly linked with the high-priority metrology interrupts to run metrology estimators which take long processing time. The period of both the metrology and control interrupts can be changed by the scientist in the initialization function of each test.

The SPHERES Core software environment was designed specifically to support multiple investigators. Satisfying the need to demonstrate formation flight control algorithms could be accomplished with a design that contemplates less threads and interfaces. Instead, the software sub-system was designed to ensure that the different scientists have access to high-level functions so that their algorithms are easy to implement.

4.2.1.4 Interface/Operations

SPHERES operations were planned specifically to create an environment which meets the MIT SSL Laboratory Design Philosophy. Figure 4.6 presents an overview of the operations plan for SPHERES. The plan consists of providing scientists with a simulation to use in-house, for quick turnaround of tests. Once the simulation demonstrates the algorithms are ready for hardware test, these are sent to the SPHERES team for testing in the MIT SSL 2D environment. The tests are sent to the ISS only after the SPHERES team has demonstrated that the tests can be run in the hardware.

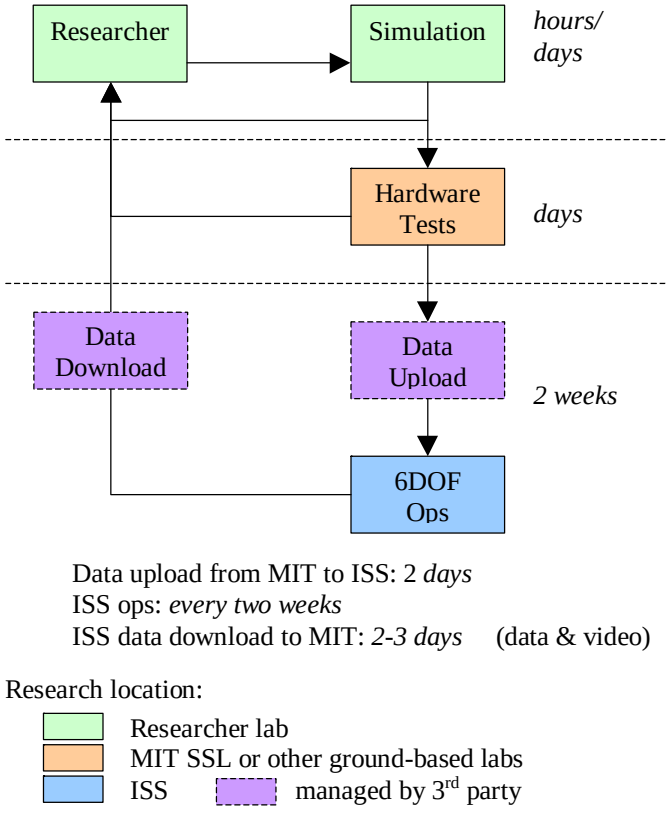


Figure 4.6 SPHERES operations overview

To implement this operational plan three interfaces were created:

- **GSP Simulation.** A simple interface to start/stop tests which provides scientists with data files to determine if an implementation is successful. The

algorithms developed in the simulation are easily portable to the SPHERES hardware

- **SSL Laboratory -** The interface for use in the MIT SSL provides detailed information on the tests being conducted as well as real-time data. The interface was designed to maximize the information to the researchers to help in the debugging and development processes. It provides the same data files that would be available from ISS tests.
- **ISS Interface -** The interface for the astronauts to conduct tests aboard the ISS was designed to meet NASA's usability requirements and to present high-level information about the tests in such a way that astronauts can make decisions on the success or failure of a test in real-time. Yet the amount of data is simplified from that of the SSL laboratory interface so that astronauts can concentrate on running the tests rather than analyzing them.

4.2.1.5 Propulsion

The satellites are propelled by a cold-gas thruster system which uses carbon dioxide as propellant. The CO2 propellant is stored at room temperature in liquid form at 860 psig, without the need for a cryogenic system. A regulator reduces the pressure to between 20-70 psig; the operating pressure may be adjusted manually prior to each test. A Teflon tubing system distributes the gas to twelve thruster assemblies, grouped in six opposing pairs. The thrusters are positioned so as to provide controllability in six degrees of freedom, enabling both attitude and station keeping control. Each thruster assembly consists of a solenoid-actuated micro-valve with machined nozzles optimized for the desired thrust of 0.125 N. The propulsion system may be easily replenished by replacing a spent propellant tank with a fresh, unused tank. The propulsion system is directly traceable to the propulsion systems of most existing spacecraft. The dynamics created by the SPHERES propulsion system directly simulate those of other thruster systems: non-linear dynamics, on/off operation, pulse width modulation or frequency modulation, and full controllability in 6-DOF. The system's bit pulse of 5ms (with an equivalent impulse bit of 0.625×10^{-3}Ns) ensures the precision necessary to operate the system at frequencies of up to 50Hz.

4.2.1.6 Structures

The primary structure consists of all of the internal and external components necessary to provide rigidity and support for the SPHERES satellite units. The primary structure functions to provide a physical base to which everything else attaches. It consists of the internal and external subassemblies. The internal subassembly consists of an aluminum frame which provides for the physical mounting of internal devices. Six internal rings comprise the main elements of the internal structure. The rings are grouped in pairs; each pair is aligned with each axis of the SPHERE. The rings fit together without a rigid connection between them. Instead, each pair is held together by four brackets; once each pair is held together, the assembly holds without connecting the rings. The external structure consists of two molded Lexan shells. The shells attach to the brackets which hold the rings together. Figure 4.7 presents CAD drawings of the internal and external subassemblies of the SPHERES nano-satellites.

The structure was designed to ensure the safety of the satellites, therefore it provides quick access only to the tank and batteries. Replacing tanks and batteries does not require any special tools nor to remove any structural elements. To safeguard all other subsystems, the structure does not allow direct access to any other internal elements of the satellites. The expansion port (described in Section 4.3.3.2) can be accessed by removing a panel attached with four screws, but it not designed for immediate access without tools.

4.2.2 Further Information on SPHERES

The previous section presents a summary of the design of the six primary SPHERES subsystems. The SPHERES hardware, software, and operational plans have undergone substantial review processes over more than four years of design and operations. The design history of SPHERES, as well as the current design, have been documented in several documents and multiple presentations. [SPHERES, 1999] and [SPHERES, 1999a] describe the design of the prototype units. [Saenz-Otero, 2000], [Chen, 2001], and [Saenz-Otero, 2002] present results obtained with the prototype units.

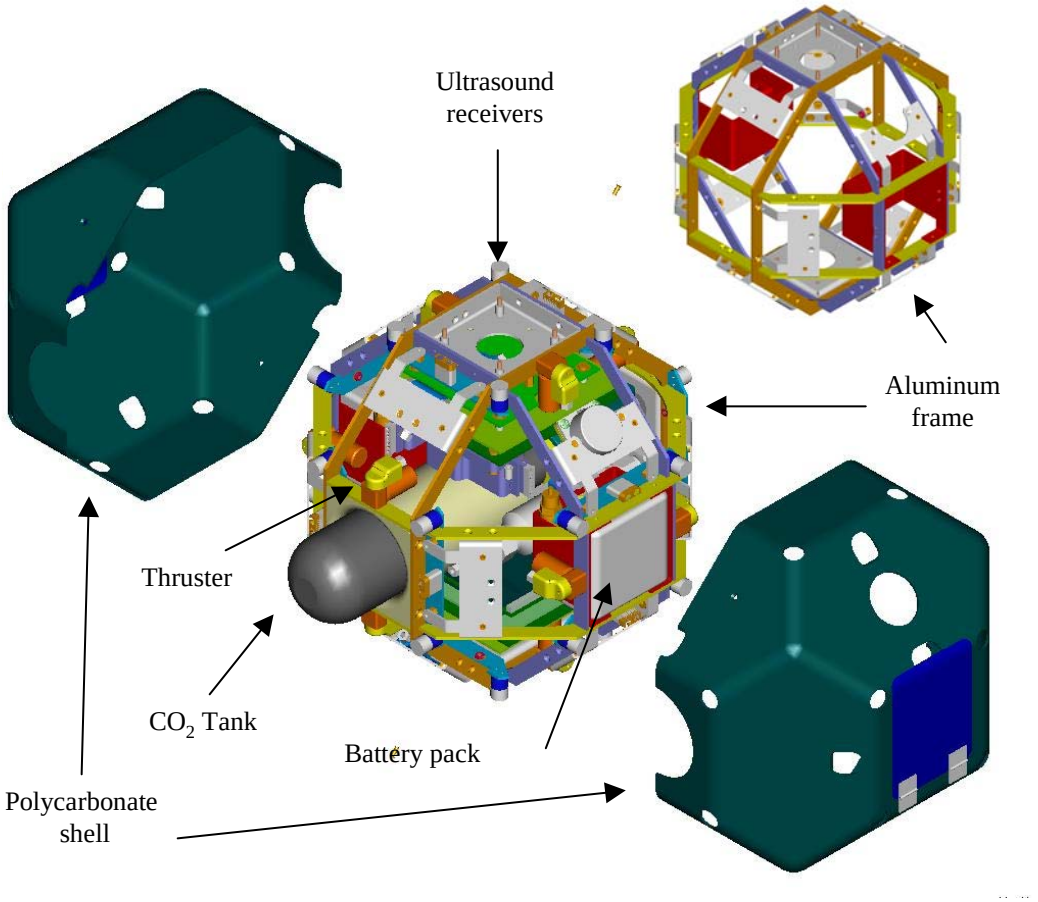


Figure 4.7 SPHERES nano-satellite structural design

[SPHERES, 1999] and [SPHERES, 1999a] are the critical design reviews for SPHERES. These presentations provide further detail on the design and operations of the flight units. [SPHERES, 2001] details the NASA safety requirements and the specific design elements which satisfy them. [Hilstad, 2003a] presents the interfaces of the SPHERES flight software available to scientists. [Nolet, 2004] and [Kong, 2004] present results from the flight qualified units in ground operations.

Due to the direct impact of the avionics, software, and communications sub-systems on the ability of SPHERES to satisfy the MIT SSL Laboratory Design Philosophy (as explained below), further detail on these sub-systems is presented in several appendices of this thesis. Appendix F presents detailed information on the avionics design of SPHERES.

The appendix present the functional block diagrams and schematics for all the electronics in the SPHERES nano-satellites, the external communications antennae, and the global metrology beacons. Appendix G presents in detail the design of the SPHERES bootloader and the flight software. Appendix H is the SPHERES communications interface document, which details the implementation of the SPHERES packets and the TDMA protocol.

The design of SPHERES contemplates the need to satisfy the goal to develop a testbed for formation flight while at the same time creating a laboratory for DSS. The following section describes how the sub-systems introduced in this section implement a wide range of features which help meet all the aspects of the MIT SSL Laboratory Design Philosophy.

4.3 Meeting the MIT SSL Laboratory Design Philosophy

The design of the SPHERES project considered each one of the design features for a laboratory, while ensuring the formation flight goal was accomplished. Table 4.4 shows the cases where a sub-system was designed specifically to meet the philosophy, to meet formation flight requirements, or both. The avionics, software, communications, and operations sub-systems most directly relate to designing a laboratory. The metrology, propulsion, and structures sub-systems were designed to meet the formation flight mission-specific goal.

The SPHERES system as a whole helps to fulfill some of the features which could not be done by an individual sub-system. Still, the avionics, software, communications, and interface/operations sub-systems implement capabilities which directly fulfill features of the philosophy. Table 4.5 cross-references the philosophy's features with those sub-systems which most influence the ability of SPHERES to satisfy the MIT SSL Laboratory Design Philosophy. The avionics system design helps to meet the lower-level features in the support of experiments and modularity & reconfiguration groups. The communications sub-system mostly supports running experiments, which in turn facilitates the iterative research process. The software and interface/operations sub-systems work at a higher

TABLE 4.4 Design for formation flight (FF) vs. design philosophy (Lab)

Sub-System	FF	Lab
Avionics		
Data Processing	✓	✓
Power		✓
Metrology	✓	
Communications	✓	✓
Software		✓
Interface/Operations		✓
Propulsion	✓	
Structures	✓	

level, using the avionics and communications systems to support the iterative research process and multiple investigators.

TABLE 4.5 SPHERES sub-systems and the design philosophy

Sub-System	Facilitating the Iterative Process	Experiment Support	Multiple Investigators	Reconfiguration & Modularity
Avionics		✓		✓
Communications	✓	✓		
Software	✓		✓	✓
Interface/Operations	✓		✓	

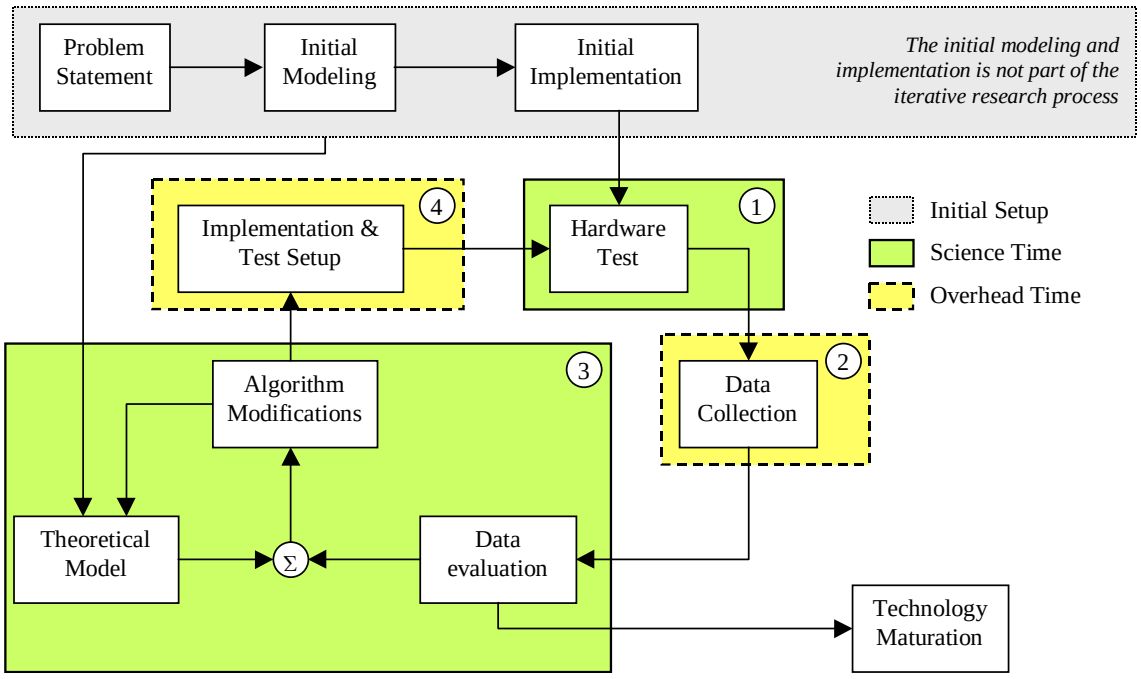
This section progressively details how SPHERES implements different capabilities which help fulfill the features called upon in the MIT SSL Laboratory Design Philosophy. Each of the next four sections concentrates on each of the four main groups of the philosophy: facilitating the iterative research process, support of experiments, support multiple investigators, and reconfiguration and modularity. Within each section, details are presented on how the implementation of a specific capability fulfills one or more of the features of the

philosophy; the sections also describe how enabling a feature sometimes required the proper integration of multiple sub-systems. At the end of the description, a summary provides a direct relationship between the characteristic and one of the MIT SSL Laboratory Design Philosophy features.

The *iterative research process* section describes the high level implementation of the SPHERES operations plan and software system which facilitate conducting science. Next the section on *support of experiments* describes several low-level hardware capabilities that were used to ensure the high-level features of the philosophy were successful. Third, the SPHERES Guest Scientist Program (GSP) and other high level features are presented to demonstrate how SPHERES allows research by *multiple investigators*. Lastly, the section on *reconfiguration and modularity* explains the low-level capabilities of the SPHERES hardware to change their configuration and create a modular system.

4.3.1 Facilitating the Iterative Research Process

SPHERES was conceived to allow for the development and maturation of control and metrology algorithms for use in formation flight spacecraft. Therefore, the iterative research process for tests performed in the SPHERES facility consists of the steps necessary to create models, develop algorithms, execute the experiments, and analyze the data to evaluate the algorithms and update them. This process must be repeatable so that the researcher can iterate during the development of the theory with confidence that environmental conditions are not changing. The specific steps identified are: initial model and algorithm development and implementation; execution in the SPHERES hardware; data collection and delivery to the researcher; analysis of the data to determine the need for further development or the achievement of maturity; and, if necessary, modification to the algorithm at different levels (either the major concepts or detailed structures such as control gain). Figure 4.8 illustrates each step of the iterative design process, as adapted from the scientific method presented by [Gauch, 2003] shown in Figure 3.1 on page 74.



Four major steps which support the iterative process:

1. Test execution (science time: allow enough time)
2. Data collection and delivery to researcher (overhead time: minimize)
3. Data evaluation and algorithm modification by researcher (science time: allow enough time)
4. Modification to tests and new program upload (overhead time: minimize)

Figure 4.8 Iterative research process for SPHERES

Figure 4.8 also identifies three different ways in which time is spent during the iterative research process:

1. **Initial development:** developing the problem statement, initial modeling, and initial implementation to prepare for the first experiments.
2. **Science time:** investigating the scientific aspects of the problem, which include the actual test time to run a significant experiment, data analysis, and development of new theoretical models and hypotheses.
3. **Overhead time:** the time necessary to collect the data and make it available to the scientist, and the time needed to implement changes in the hypothesis and start a new test with the updated algorithms.

The design of SPHERES concentrates on the four main steps that support the iterative research process, identified in Figure 4.8 by numbers within circles: first, on providing scientists with the correct amount of time to run tests (science time); second, minimizing

data collection and delivery time (overhead time); third, providing enough data evaluation and model refinement time (science time); and fourth, enabling easy modifications of the algorithms (overhead time). The initial theoretical analysis and implementation is considered a constant outside of the scope of the iterative research process; these issues are addressed by other features of the design philosophy, such as the ability to support multiple investigators.

The goal of the SPHERES facility is to provide sufficient science time, while minimizing the overhead time. A successful facility allows researchers to run tests for long-enough periods of time to return valuable data. If the time to perform experiments is too short, then the amount of useful data will be reduced; short experiment time may even prevent a test from completing, in which case the overhead of restarting a test becomes substantial. The time for step two should be minimized, meaning that the time to collect the data and make it available to the scientist in a useful format should be as short as possible. The third stage, where the researcher analyzes the data must be more flexible. This time should not be so short that the researcher is unable to perform careful data analysis; nor should it be so long that the scientist is unable to effectively track the evolution of the process or meet mission deadlines. It would be preferable for this time to not be fixed, but rather allow the scientists some leeway to ask for more time if necessary, or potentially to speed up the process if new algorithms are created quickly. The time of the fourth stage, the implementation of changes and setup of new tests, should be minimized. The time must be such that the researcher will not loose interest on the next test, and will remember all the changes performed in the last iteration along with their rationale. In the case of SPHERES this process involves the creation of a new program and its delivery to the appropriate location for upload to the satellites.

SPHERES must allow researchers access to each step of the iterative research process with efficiency, allowing the algorithms to be developed not only correctly but within a reasonable amount of time. For this purpose, the team considered not only the design of

the testbed itself, but also the resources available within the ISS that should interface with the facility to achieve this goal.

To facilitate the iterative research process, i.e. to minimize overhead time and maximize science time, SPHERES implements the following capabilities:

- Multi-layered operations plan
- Continuous visual feedback
- Families of tests
- Easy repetition of tests
- Direct link to ISS data transfer system
- De-coupling of software from NASA safety controls

The implementation of these capabilities are detailed next.

4.3.1.1 Multi-layered operations plan

The SPHERES operations elements that benefit the iterative process consist of three main elements: the Guest Scientist Program (GSP) simulation, ground based facilities, and ISS operations. Figure 4.6 on page 111 shows an overview of the SPHERES operational modes that enable iterations. As seen, the longest cycle, when the experiments are conducted aboard the ISS, completes in a matter of a few weeks. The benefits of each of these elements is described below.

Iterations with the GSP Simulation

The GSP Simulation allows remote researchers to develop their algorithms in house at their own pace. Figure 4.9 illustrates the iterative research process of the GSP simulation environment. The only overhead related to the simulation is the need to convert all of the researcher's algorithms into C code and make it fit within the SPHERES software Application Programming Interface (API). While the initial time spent on converting the code to C may not be negligible, it is a necessary step to operate on the SPHERES hardware. Therefore, the initial time spent to convert the code will be useful in the long-term, as that

code will serve as a base for tests that may ultimately be performed aboard the ISS. The simulation allows multiple iterations of a technology to be accomplished in a few hours, after the initial overhead to translate the algorithms (expected to be a period of a few days).

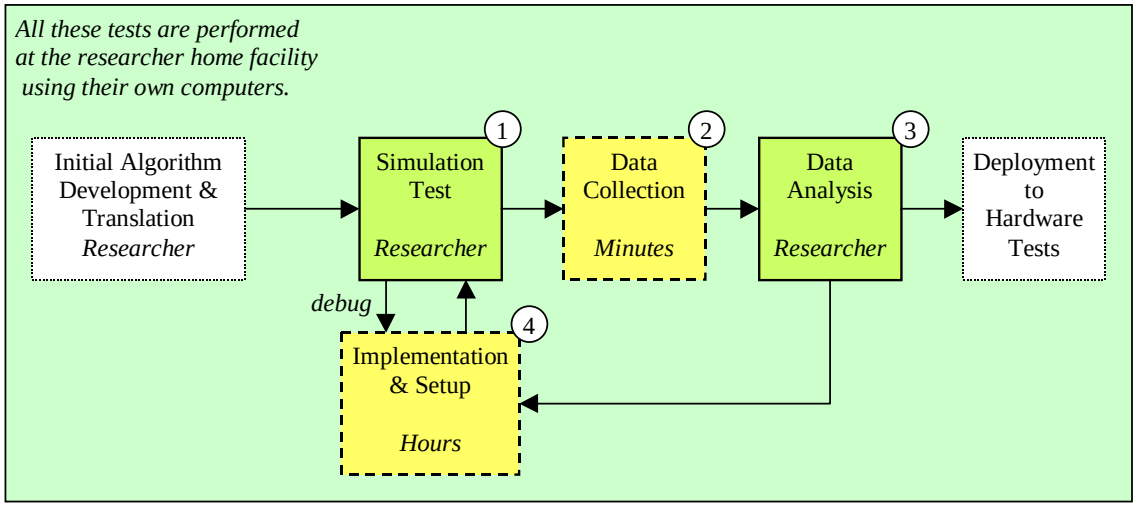


Figure 4.9 GSP iterative research loop

Once the code has been adapted to the SPHERES API, the researcher may run tests using the simulation, which provides the same data that a hardware test would provide. The data is augmented with the state of each satellite as calculated by the simulation independent of the metrology algorithms in use by the researcher. The flight-style data ensures that iterations in the hardware will contain the necessary telemetry; the simulation calculated state serves as a truth measure to determine the success of the researcher's algorithm within the simulation.

Iterations at the MIT SSL

The MIT SSL provides researchers with a low-stress environment where 2D (3DOF) tests can be performed. In this environment the researcher can easily modify tests and programs in multiple development stations, and the overhead to reload a program (approximately 2-10 minutes), does not present a considerable delay. While at the MIT SSL, the researcher

will spend most of their time evaluating the data, running tests, and modifying tests to improve their performance. Further, the researcher need not be present physically at the MIT SSL. The MIT SPHERES team will match on-site students with partner researchers in such a way that the researcher remains in their main location while the member of the SPHERES team will conduct the experiments and relay data to the researcher as needed. Because the team members are fully proficient on the operations of SPHERES, this process speeds up the iterations by allowing the researcher to concentrate on their science, instead of the SPHERES operations.

In the ground-based SSL facility, the first step of the iterative process, running the actual test, is limited by the capabilities of the equipment that allows 3DOF operation and the consumables of the testbed, regardless of the operating scenario. The consumables of the air-carriages that support the satellites last up to 20 minutes. The gas on the satellites, in ground operations, lasts approximately 20-30 minutes. Therefore any continuous test is limited to 20 minutes. While all the ground operations so far have required test times of less than 10 minutes, this constraint remains a valid hard constraint on the iterative research process steps.

Since researchers may or may not be present at the MIT SSL to run their experiments, there are two possible iteration time lines for this environment. When the researcher is present at the MIT SSL, the overhead time is minimized greatly, requiring only minutes to obtain the data and to setup experiments. On the other hand, unless the researcher is based out of the MIT area, the time in between iterations will be restricted by the costs of remaining on site. Conducting tests on-site has proven most useful when the researchers are on the last steps of their design and wish to only optimize the last details of their algorithm with a quick turnaround between tests. Figure 4.10 illustrates the on-site iterative research process.

The other operating scenario is when the researcher operates remotely and is supported by a member of the SPHERES team. Figure 4.11 illustrates the off-site iterative research pro-

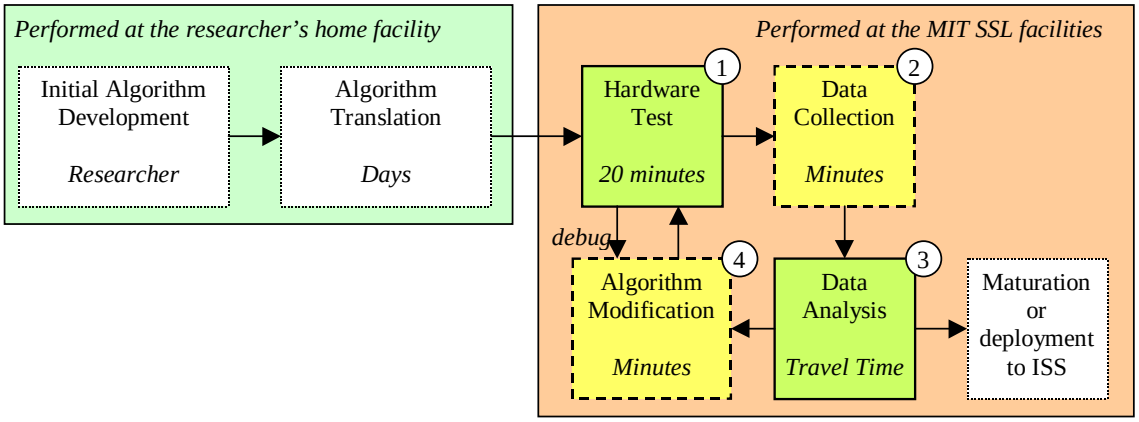


Figure 4.10 MIT SSL on-site iterative research loop

cess. In this case the overhead time to collect data and setup experiments increases to days. The data transfer can usually take place within hours, but the uploading of new programs requires the SPHERES team member to compile the program for use in the hardware; this process can take up to a few days. On the other hand, the researcher has practically unlimited time (up to months, if so desired), to analyze the data and produce new or modified algorithms. The GSP Simulation enters the loop once more, as scientists will test their algorithm modifications with the simulation prior to sending new programs to the SPHERES team.

Iterations aboard the ISS

Operations aboard the ISS tie in all the steps of the SPHERES operations procedures: GSP simulation, ground facilities, and ISS operations. Any tests to be performed in the ISS must prove capable to operate both on the GSP simulation and the SSL before the SPHERES team will allow delivery to the ISS. While tests at the SSL are not expected to perform all maneuvers expected from the 6DOF environment of the ISS, all tests will be checked for errors that could affect the operations of SPHERES; in those cases where the success of the testbed cannot be shown in the 3DOF SSL environment, they will be expected to perform correctly in the GSP simulation and, at a minimum, successfully load

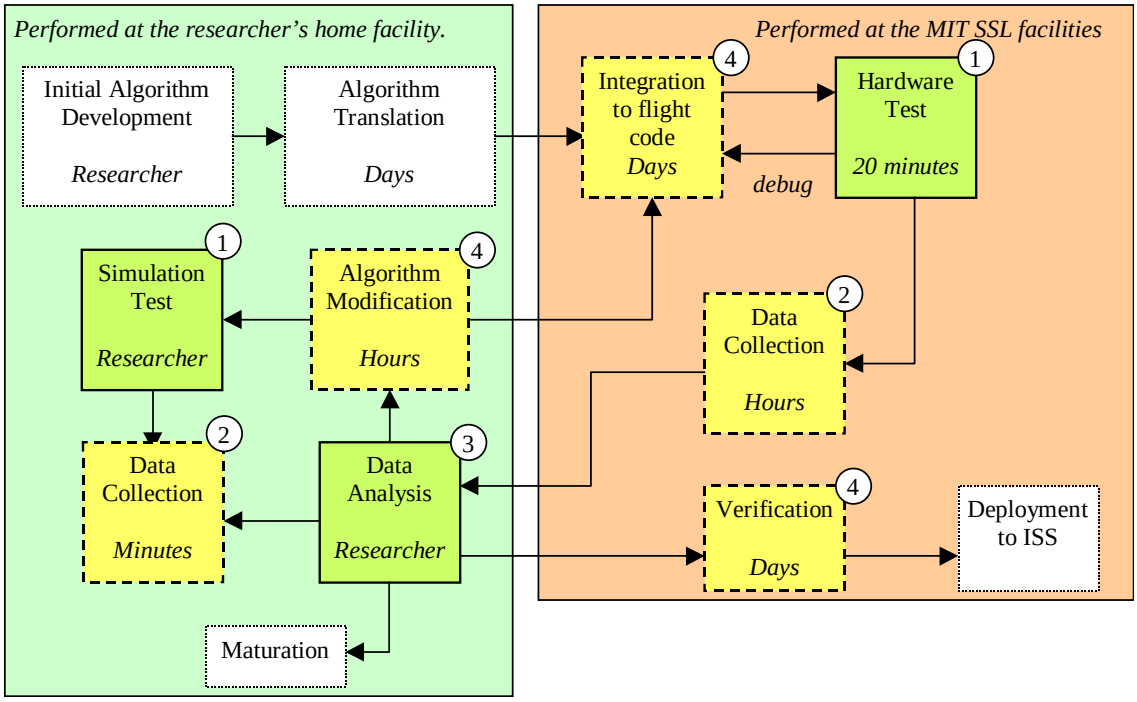


Figure 4.11 MIT SSL off-site iterative research loop

and run on the C6701. Therefore, the iterations aboard the ISS will include the overhead of both the GSP process and the SSL operations.

The ISS also adds other overhead times of importance to the iterative research process. To operate aboard the ISS the SPHERES team must interface with NASA via our payload sub-contractor, Payload Systems Inc. (PSI), and the payload sponsor, the Department of Defense (DoD) Space Technology Program (STP). While the software, as described above, presents no safety-critical items that will require NASA verification, it must be uploaded to the ISS via both PSI and then STP, followed by the NASA ISS office. This process will require the SPHERES team to have software ready for upload several days in advance of the up-link. Once NASA has received the programs for upload, the up-link to the main ISS server will occur within one day. The astronaut can then copy the program from the server to any one of the general purpose laptop computers aboard the ISS prior to the operating session.

Data download will also have added overhead, since both PSI and STP must be involved. Further, the data down link from the ISS is not necessarily real time. For the majority of the SPHERES operations NASA has indicated a one day download time of raw data (including astronaut questionnaire and feedback), and a two to three day video download. The data must then pass through STP and PSI before reaching the SPHERES team and/or researcher (in some cases PSI may send data directly to the researcher).

As explained in Chapter 2, astronaut time is a precious resource aboard the ISS. SPHERES has been allocated a fixed time for operations over a period of six months. The SPHERES team decided to allocate the time in intervals of two hours of operations every two weeks. This time will be fixed by NASA once operations start (although SPHERES operations may be preempted by other NASA activities). Therefore, the minimum time for data analysis and algorithm modification for researchers will come in intervals of two weeks. A scientist may decide to have the time between iterations be every two, four, even six weeks or more depending upon their needs, but not less than two weeks.

While the total time per session is two hours, one must recall that the limiting factor for each test are the consumables. Specifically, the available propellant in each tank is estimated to last for up to 30 minutes (depending heavily on controller usage). Therefore, each test can be at most 30 minutes long, assuming conservative gas consumption and that a new tank was used at the start of the test. Longer tests may be possible for minimal gas consumption. At that point, the limiting factor becomes the batteries, which last up to two hours, regardless of the maneuvers of the satellites.

At this point it is important to note that, while in ground operations the time to upload a program to the SPHERES satellites was considered negligible, this can no longer be ignored in the ISS. Uploading a program takes up to five minutes per satellite that must be programmed. Therefore, for a three satellite test, the overhead will be up to 15 minutes; this is a considerable amount of time out of the two hours allocated per session. Therefore,

the SPHERES operational plans call for no more than two programs (each with multiple tests) to be uploaded per session.

Figure 4.12 on page 127 presents the ISS iterative research process graphically. The process starts at the researcher facilities utilizing the GSP simulation. The total overhead at this point is in terms of hours to run enough simulations and debug software. The process continues with the delivery of the software to the MIT SSL for integration into a flight package. This process will add several days of overhead to ensure the software is ready for delivery. Once validated, the program is sent to the ISS via PSI/STP/JSC, for a total overhead of approximately two days. The astronaut, once scheduled to operate SPHERES, copies the program to a local laptop and loads the program, for a total overhead of approximately 15 minutes. A cycle of tests follows; multiple tests are run, guided by the astronaut's decisions and the operations plans provided by the scientist. The data collected on the ISS laptop is copied to the main ISS server immediately after the session in a matter of minutes; the data is made available to STP/PSI by JSC within one day. Within another day the data reaches the researcher, for a total data download overhead of approximately two days. The video is downloaded from the ISS in approximately two or three days, and made available in digital format to STP/PSI in another few days. In total, the video of a test session is expected to be available within one week of the test session. The researchers are expected to operate on a four or more week cycle to allow sufficient time for data analysis and algorithm modifications.

> *A multi-layered operations plan allows scientists to perform research iterations in-house, remotely and locally at the MIT SSL, and remotely at the ISS. The iteration period ranges from hours to a flexible 2-week schedule for operations aboard the ISS.*

4.3.1.2 Continuous visual feedback

A major obstacle to maturate technologies via simulation is the inability to fully understand the dynamics of a system and visualize them properly. SPHERES provides research-

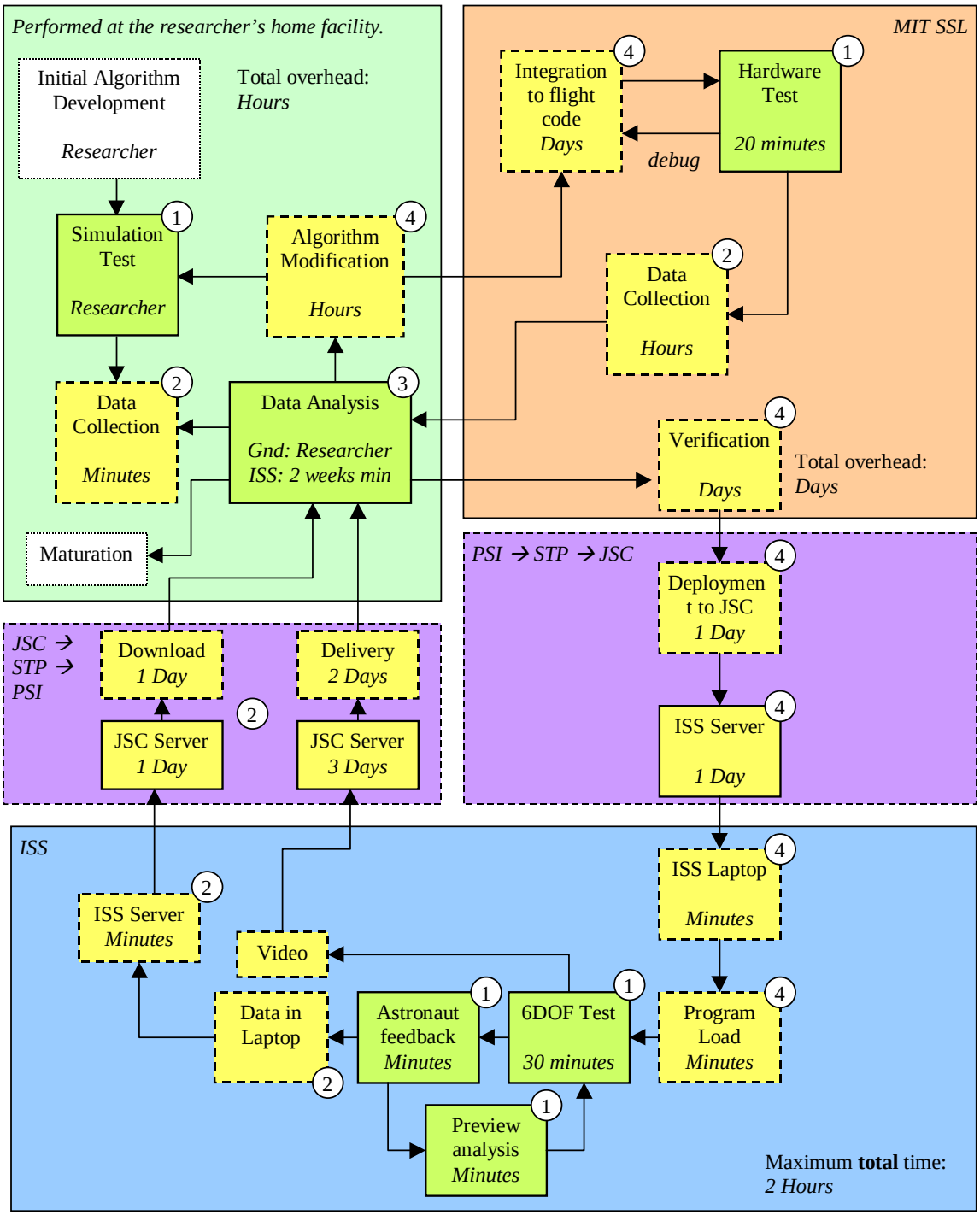


Figure 4.12 ISS iterative research process

ers with a physical system where they can carefully study the behavior of the satellites. When the researcher is present, they can directly identify the dynamic behavior of the satellites, which allows them to quickly determine the success of a test. Further, they obtain much more insight into the actual three dimensional behavior of the units, even if conducting tests in the 2D ground facilities. When SPHERES is operated remotely of the researcher, the facility always provides researchers with two visual feedback elements: a human is always present to evaluate tests in real-time, and video will always be available. In many cases video is available from multiple angles (including ISS video), which can potentially be used for data analysis.

The ability of SPHERES to minimize the data collection overhead time is directly related to the availability of visual feedback and video in all locations. The physical operations provide the researchers with immediate feedback to make rough determinations of success or failure of the experiments. Since the SPHERES facility was designed for the development of dynamics algorithms, in many cases it will be clear from the video when an experiment succeeds or not, since specific motions will be expected. Using video the scientist can then determine which data sets to investigate further, saving time by not having to analyze every single data set.

> *Continuous visual feedback by humans allows scientists to reduce the data collection overhead time by filtering useful experiment runs.*

4.3.1.3 Families of tests

The SPHERES software enables researchers to run a full family of tests with ease. The SPHERES software consists of programs that contain multiple tests. At any point in time only one program can be loaded into a SPHERES satellites. The program can be changed easily with the bootloader, described in Section 4.3.4.7 below. But changing a program does have a two to five minute overhead, which, as described above, is not negligible in the ISS environment. Therefore, to minimize the overhead in starting experiments, SPHERES allows each program to perform multiple tests.

Theoretically the different tests in each program can be completely independent of each other; in practice the SPHERES team attempts to maintain similarity between the different tests in each program to minimize the size of the program and to ensure that subsequent tests make sense operationally. In the case of a controls problem, for example, a program may contain multiple tests of the same algorithm with different gains for the controller; but the tests could also sequentially build on the controls problem, adding steps with each test. The ability to run these families of tests sequentially, without any substantial overhead, allows for different parts of an algorithm to be tested individually and then collectively, until the full algorithm is demonstrated.

Individual tests can be further divided into maneuvers. Each test can contain one or more maneuvers, allowing the researcher to further divide the test and identify up to what maneuver a test performed as expected. As opposed to tests, which can be started in any order, the user does not have control to start a test at a specific maneuver; tests must always start with the first maneuver. But the software architecture allows maneuvers to be shared among tests, and a researcher can create tests with overlapping maneuvers to test different parts of an algorithm.

Figure 4.13 illustrates the structures of programs as implemented in the SPHERES software environment, and describes an example for the "checkout" program. This program tests the different sub-systems of the SPHERES satellites independently of each other, therefore demonstrating the ability of one program to perform substantially different tests. Operationally, though, the checkout program is congruent, as the operator will clearly identify each sub-system with a test. The example also shows how maneuvers can be used to simplify test design. In the checkout program the "global metrology" test, which checks the functionality of the ultrasound receivers in the satellites, uses maneuvers to keep track of which satellite face is being tested, rather than having to create special telemetry.

The implementation via families of tests and ability to change programs further facilitates the iterative research process by effectively allowing the software to be modified at any

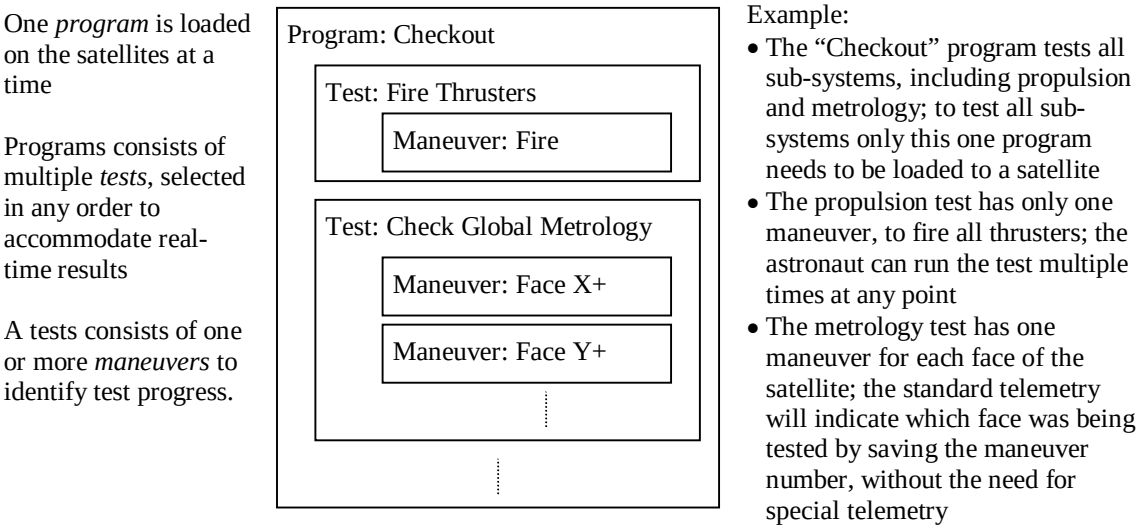


Figure 4.13 SPHERES programs composition

level. When considering steps three and four, algorithm modification and test implementation and setup, this design gives the scientist several advantages. During the algorithm modification time the scientist can decide to review only specific maneuvers, combine tests, or even redefine the program completely. There is no overhead in terms of reloading the program to choose any level of modification (albeit, small changes result in the need to load the full program again). By allowing the scientists to modify their algorithms at any level, SPHERES maximizes the time scientists can spend in re-defining their algorithms, rather than implementing them. The software sub-system ensures that the iterative process is not slowed down whether small or large changes are needed.

> *Running families of tests minimizes the overhead to start a test and allows multiple parts or types of algorithms to be tested in one session.*
> *The layered program structure enables both small and large changes to the algorithms.*

4.3.1.4 Easy repetition of tests

Chapter 3 reviewed two essential concepts in conducting research: the scientific method and the design of experiments. The operations plan presented above works to fulfill the

need of the scientific method to iteratively improve a hypothesis. The design of experiments, on the other hand, closely relates to the ability to conduct the correct number of tests, with a number of variables, to demonstrate the statistical viability of a hypothesis. This concept lies within step one and four of the iterative process presented in Figure 4.8 on page 118: minimize the time to setup a test to maximize the science time. To accomplish this goal, the design of SPHERES ensures that it is possible to repeat tests with ease.

This repetition of tests takes place within a specific program; the times to minimize are the ones involved with setting up the units for a test and commanding the start of the test. Starting a test with SPHERES consists of four simple steps:

1. Check battery and gas pressure: visual feedback of both available battery (low battery indicator) and gas tank contents (estimated use) allow this check to take place in seconds.
2. Enable the satellite (mandatory only in the ISS): for safety reasons the satellite must be enabled prior to any actuation. This process requires the astronaut to depress the enable button for more than one second.
3. Position the satellite correctly: this is potentially the most time-consuming part, since it is desired that all tests of the same algorithm start with similar conditions. Still, the small physical size of the satellites (both mass and volume) allows easy manipulation so that similar starting conditions can be achieved. Tests in the ISS by astronauts, with objects of similar size to the SPHERES satellites, have demonstrated that preventing drift of one unit while other units are re-positioned does not pose a challenge.

   In the case of ground tests, positioning the satellites also requires positioning their air carriages correctly and turning on the gas for the carriages. This process takes only a few seconds for one or two SPHERES, although it can pose an operational challenge for three or more units. Ground tests have shown that the air-carriages do drift once floating, and therefore require more human attention to ensure a successful start.
4. Command test start: the ground-based interfaces command a test start with the simple press of one key. The ISS interface requires two steps to satisfy safety requirements. (Both interfaces are described in further detail below).

The total time to start a test is less than one minutes in ground stations when using up to two satellites, and approximately one to two minutes for the ISS. When using three or more satellites in ground facilities it may take up to five minutes to start a test, depending

on the conditions of the surface where the carriages are floating and the amount of drift experienced.

Another important aspect of repeating multiple tests easily is the ability to stop a test that has gone wrong and reset the facility for the next test. SPHERES allows multiple ways to stop a test and setup the facility ready for a new test. The software can restart a test in multiple scenarios. The researcher can program special conditions which cause a test to automatically end, leaving the satellite ready for the first step of setup. The software implementation ensures that a test stop will not affect the behavior of the rest of the system. This feature allows tests to be run only as long as necessary, allowing tests to be stopped if they fail in an obvious manner, while ensuring the data is safely archived for post-examination. A remote restart command, which causes the units to re-start in a manner equivalent to cycling the power, was implemented in case the satellites are out of reach. The satellites' control panel allow the scientist to disable the current test, which does not perform an actual reset of the avionics, or to reset the unit completely, also equivalent to cycling power. Through these four different reset scenarios SPHERES allows a scientist to both stop tests without affecting the state of the avionics or to fully cycle the satellites for a clean start.

The ability to physically restart a test is not useful unless it is possible to identify which set of data corresponds to each test. SPHERES transmits multiple telemetry packets which clearly identify each run of a test, regardless of whether a previous test initiated, ran, or concluded as expected. The system is further enhanced by the different operator interfaces, all of which save all raw data, allowing for error-correction in post-test analysis.

SPHERES implement these features by creating a high-level state machine as part of the basic SPHERES software and utilizing the external watchdog of the avionics system to force resets. Figure 4.14 illustrates the state machine implemented by the software. Four states are implemented: boot time, which initializes the satellites and load the current program; idle, which does not perform any actuation and which is not running any user

threads (see figure Figure 4.5 on page 109), but which sends the "state of health" (SOH) information once a second, which saves the current state, the satellite on-time, and tank usage, among others; the "ready" state is an intermediate step which allows a test start command to be acknowledged, but which otherwise is not functionally different from idle; the running state, which starts after a command has been acknowledged, first runs the test-specific initialization code, and then start the user threads. If the researcher programmed their test-specific initialization code correctly, then whenever the satellite starts a new test the conditions of the satellite will be the same. Note that the SOH packet downloads full test information when a test is running, so that each test can be clearly identified.

> *The SPHERES state machine implementation, coupled with hardware reset capabilities, ensures that tests can be stopped and started with minimal overhead.*

4.3.1.5 Direct link to ISS data transfer systems

While both video and experiment data are easily available in ground testbeds through local computers and camcoders, the iterative research process can be greatly disturbed by lag of data transfer while operating aboard the ISS. Previous experience taught the SPHERES team to minimize the need for communications hardware outside of the available ISS resources, such as using our own hardware for storage of data and/or requiring transfer of physical items such as CD’s or video tapes. Therefore, the SPHERES maximizes the use of ISS resources to minimize the amount of time spent transferring data between systems, while at the same time balancing the fact that this makes the collection of SPHERES data and video dependent on a third party (NASA).

The SPHERES facility utilizes an ISS laptop as the control station. The SPHERES interface runs directly on that laptop; all the data received by the laptop are saved directly to the laptop (at the end of an ISS test session a single compressed file collects all the data). NASA has provided the SPHERES team with a shared drive for use, which means the SPHERES data files will have a direct link to the main ISS server. Under normal opera-

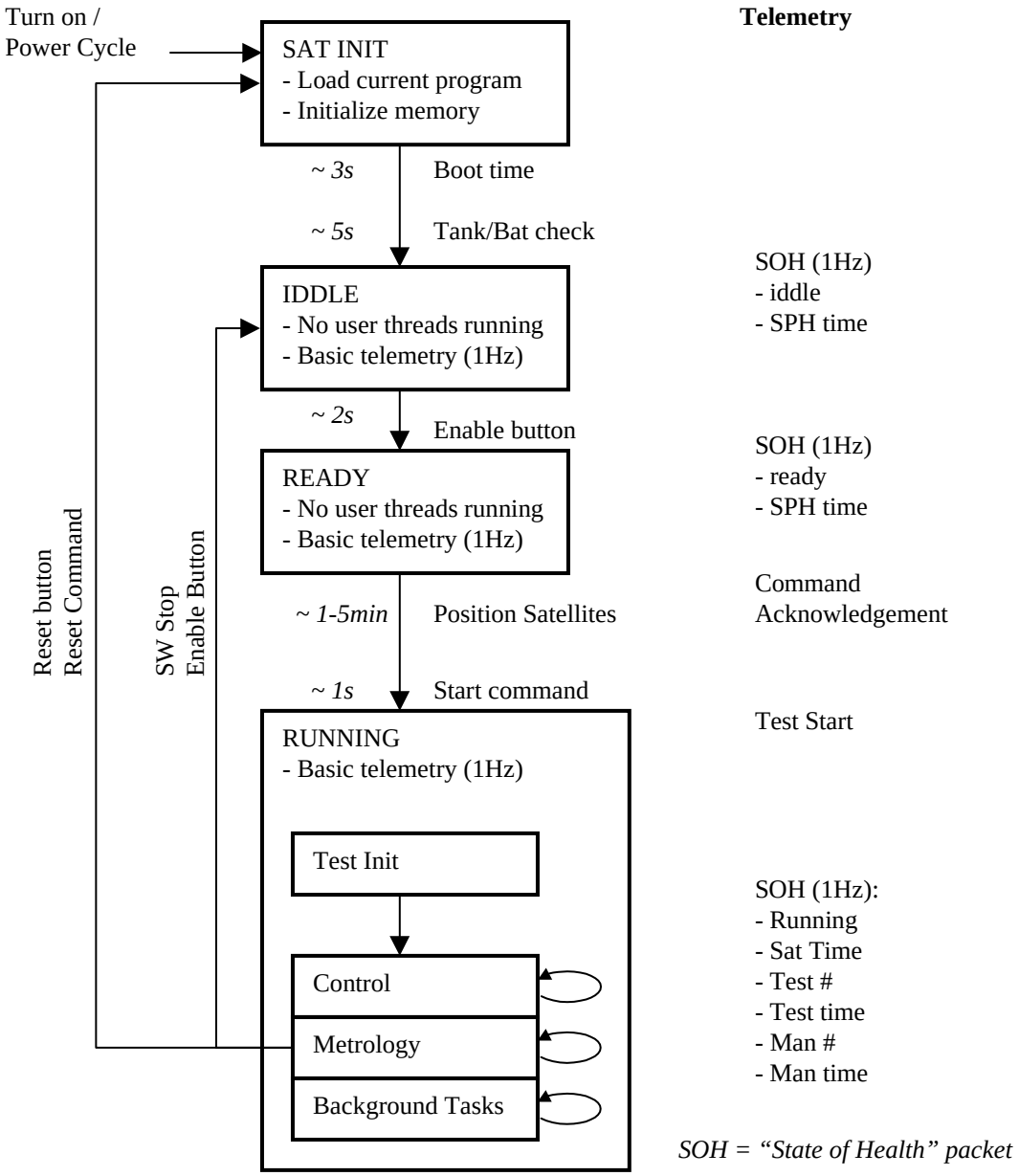


Figure 4.14 SPHERES satellites initialization

tions the astronaut will not need to transfer files manually, the operation occurs automatically. The data from the main ISS server is downloaded every day by JSC. At that point the team depends on the JSC staff to forward the data to the SPHERES team; this usually occurs within 24 hours.

SPHERES also utilizes the ISS video capabilities for both capture and download. While this process can take longer than data download, it is still faster than any custom solution. Utilizing the ISS existing hardware prevents the need for further mass/volume if SPHERES custom video equipment were required. To obtain reasonable download times the ISS data system would still have to be used for video download, possibly slowing down the transfer of data overall. Because downloading server data and video are separate processes at NASA, the use of the ISS video system decouples the download of the two sets of feedback.

As a result SPHERES does not require of any data transfer between custom media and does not need to wait for the physical return of any data storage facility. The only SPHERES specific communications hardware is the STL (Satellite-to-laptop) transmitter, which connects directly to the ISS laptop.

> *A direct link to the ISS data system minimizes the time to collect microgravity data and requires no transfer of physical media.*

4.3.1.6 De-coupling of software with NASA safety controls

Throughout the development of the SPHERES facility great care was taken to ensure that the software was not part of NASA's safety controls. As explained above, the software facilitates the iterative research process by allowing scientists to change their implementation at many levels. If the scientist discover that only small changes are needed, but the SPHERES software had to go through NASA safety reviews to be implemented, then the overhead for small changes would be disproportional to the changes. To ensure that the implementation step during ISS operation remains reasonable, it was essential for the SPHERES software to remain independent of NASA safety controls.

To achieve this goal the hardware sub-systems provide the double and triple redundancy required by NASA. The NASA safety reviews concentrated on the propulsion, power, and communications sub-systems, on the fact that the SPHERES satellites are free-floating in

the ISS, and on the noise levels produced by the satellites and metrology hardware [SPHERES, 2001].

The propulsion system provides hardware relief valves to ensure that the unit is not dependent on the software to vent the unit in case of pressure build-up. Further, even though not required by the NASA safety review panel, the propulsion hardware driver circuits defaults to an off state, such that resetting the SPHERE will always close the solenoids and stop any firings, independent of test-specific software. Lastly, the amount of thrust exhibited by the satellites is such that an astronaut can hold a satellite until the gas tank depletes without any hazards. The power system provides hardware current limiting devices directly in the battery packs (redundant), a magnetic circuit breaker, and always operates below 32V. These specification mean that the software can never cause a safety hazard situation due to electrical power. The communications system defaults to idle mode, and requires initialization from the software. Further, the reset signal is sent independently to the communications hardware, ensuring that a reset forces communications to stop regardless of the state of the software. Lastly, the dynamics of each satellite were designed so that even in the case of a software failure any collision is under the limits specified by NASA and the noise produced by the satellites was designed to be below NASA requirements during all types of operations, including excessive thrusting. These measures ensure that once a researcher decides to change a program, this can be uploaded to the SPHERES project without any delays due to software reviews.

A further benefit of the lack of safety checks for the ISS is their applicability to operations in multiple NASA locations, including the KC-135 Reduced Gravity Airplane. The software cannot cause any hazards by itself, and therefore does not require verification. The lack of safety controls in the software ensures that such lag does not exist, and that the iterative research process is only affected by the availability of communications links to and from the facility.

> *De-coupling all safety controls from the software minimizes the overhead to deploy new algorithms and allows scientists to program algorithms that push the limits of the theory without delaying the iterative research process.*

4.3.2 Support of Experiments

The need to repeat experiments in an efficient matter is an essential part of the iterative research process. As presented in Chapter 3, the "support of experiments" features directly affect the ability to enable the iterative research process, but they go into further detail on how to achieve efficient tests than the high-level feature of "facilitating the iterative research process". The last section presented the high level operations plan and other special characteristics of SPHERES which facilitate the iterative process over all. This section will present in more detail the several specific functions of SPHERES that directly affect its ability to run an experiment correctly and efficiently.

The following functionalities of SPHERES enable it to support conducting experiments:

- Data Collection and Validation Features
  - Layered metrology system
  - Flexible communications: real-time & post-test download
  - Full data storage
  - 32 bit floating point DSP
- Redundant communications channels (reliability)
- Test management & synchronization (repeatability)
- Location specific GUI’s (human observability & manipulation)
- Re-supply of consumables (repeatability, supporting extended investigations, risk-tolerant environment)
- Operations with three satellites (reliability, risk tolerant environment)
- Software cannot cause a critical failure (risk tolerant environment)

4.3.2.1 Layered metrology system

Chapter 3 calls for the following requirements on data collection (among others):

- Ensure data accuracy and precision scalable to the final system

- Ensure observability of the technology

Developing the SPHERES environment presented a major challenge in the development of a valid metrology system. Not only was a 6DOF measurement system for use inside the ISS was not available "off-the-shelf", but the design of SPHERES called for it to be a development testbed for metrology algorithms itself. Therefore the metrology system had to provide scientists with both the necessary data and computation power to both develop new metrology algorithms and to trust the data if they use the SPHERES provided procedures.

To obtain the accuracy and precision scalable to the final system the SPHERES metrology system utilizes a two-layer implementation. High-frequency accelerometers and gyroscopes capture the inertial motion of the satellites. The selection of the COTS accelerometers and gyroscopes was driven by the need to provide accurate data for the expected motion of the satellites using the cold gas thrusters. With a thrust of approximately 0.125N and a mass of 4kg per satellite, each thruster would cause an on-axis acceleration of approximately 0.03125 m/s^2; the thrusters are located approximately 0.125m off axis, providing a torque of approximately 0.004Nm, which equates to a rotational acceleration of 0.25rad/s^2. Therefore, the accelerometer must measure 3.2mg's, while the gyroscope range was selected as ±50°/s (~0.9rad/s) since the expected thrust periods are no longer than one second (which results in approximately 30°/s with two thrusters firing on axis). Table 4.6 summarizes the satellite dynamics under thruster actuation.

TABLE 4.6 Satellite dynamics under actuation

	Value
Single thruster force	0.125N
Minimum opening time	10ms
Acceleration	0.03125m/s^2 (~3.2mg's)
Rotational speed (minimum impulse)	0.25rad/s^2

The gyroscope selection was relatively straight forward, as the range of rotation rate was easily available from COTS equipment and the thrusters have a large enough level arm to easily excite the single-bit resolution of a wide range of gyroscopes. The final selection took place for their size and small drift which allows rotational control of the units for extended periods without the need for the global metrology system. The gyroscope specifications are presented in Table 4.7.

TABLE 4.7 Gyroscope specifications (BEI Gyrochip II)

Specification	Value
Input range	±50°/s
Scale Factor	30mV/(°/s)
Bias stability (<100s)	0.05°/s
Bandwidth	50Hz
Sensitivity (12bit A2D)	0.0407(°/s)/bit

The selection of the accelerometer presented a bigger challenge, since the thrusters linear acceleration is in the milli-g's range. Therefore the selection had to trade-off between selection accelerometers which would provide feedback on external disturbances (such as collision of two units at dock time or forces by humans) or from the thruster firings. The final selection gave priority to measuring thruster actuation. The accelerometer will saturate whenever external forces, which will be much larger than the thruster forces, are applied; the saturation of the accelerometers can be used as a binary method to detect external forces, even if not to model them. The characteristics of the accelerometer selection are presented in Table 4.8.

Testing of the gyroscopes and accelerometers in the KC-135 reduced gravity airplane showed the ability of the sensors to detect the thrusters. Figure 4.15 plots sample data from the micro gravity tests. The data are from testing firing thrusters number zero and one, which produce acceleration on the X axis and rotation about the Z axis. The top plot presents the accelerometer readings; the bottom plot the gyroscope data. While in an ideal

TABLE 4.8 Accelerometer specifications (Honneywell Q-Flex QA-750)

Specification	Value
Input Range	±30g
Scale Factor	1.33mA/g
Resolution	1μg
Bandwidth	300Hz
Amplifier Gain	2000Ω * 40
Sensitivity (12bit A2D)	.011473 mg/bit
Effective Range	±24mg

situation the accelerometer data would show a square wave as a thruster turns on and off, the plot shows the effects of placing the accelerometers off axis. The sensitivity of the accelerometers is so low that they measure not only the thrusters, but also the centripetal acceleration due to the offset. Following the plot piece wise we see the following parts:

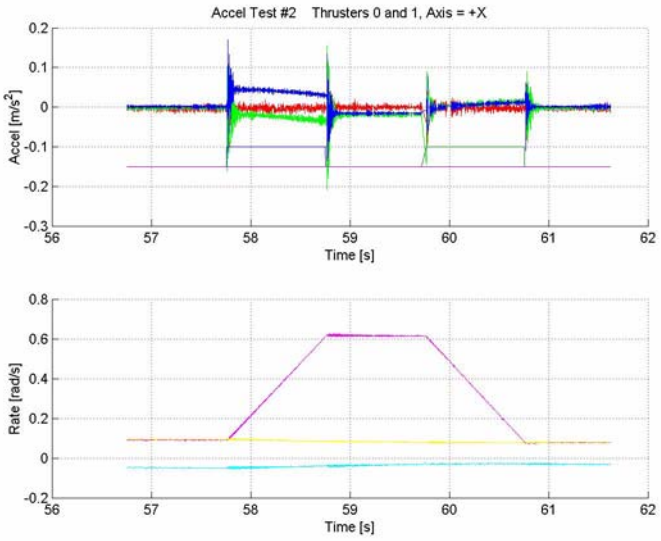


Figure 4.15 Accelerometer and gyroscope measurements in micro gravity

56.7-58.7s: initially at rest; no acceleration or rotation

57.7-58.7s: Thruster 0 turns on, causing +X acceleration (top lot blue line) and +Z (bottom plot purple line) rotation. Note that as the rotational speed goes up,

the off-axis effects cause the +X acceleration to go down. Also note that thruster zero causes a small effect on the Z axis accelerometer.

58.7-59.7s: All thrusters are off again; note that the accelerometers do not return to zero because of the rotation of the unit.

59.7-60.7s: Thruster one turns on, stopping the rotation of the unit. The effect of the thruster is not seen much in the accelerometer in absolute numbers because the acceleration due to rotation was similar to that caused by the thruster; the effect is seen my the difference in accelerometer readings between the past period and this period.

60.7-61.7s: The units return to rest; with no more rotation the accelerometers return to their zero state values.

A simplistic model of the SPHERES satellites which does not account for the coupling between rotational speed and the accelerometer readings would not be able to use the accelerometer data. Therefore, while the accelerometers can provide full observability of the system, this is only true if the correct model of the satellites is used.

A low-frequency ranging system uses a combination of infrared and ultrasound signals to measure distances using the "time-of-flight" of the ultrasound signals and provide position and angular direction in 6DOF within a pre-defined operating area. Five external transmitters are used. An infrared pulse (treated as instantaneous) commands the start of ultrasound pulses and marks time "zero". The transmitters pulse one at a time in 20ms windows (allowing up to 6m of travel by the ultrasound). Each satellite uses 24 ultrasound receivers, four per face, allowing scientists to use the information for both triangulation of position and differential measurement for angular direction. Scientists can use the direct measurements or filtered data as best fits their application. Figure 4.16 illustrates the raw distance measurements of the global metrology system. The system provides resolution of ±5mm and ±2° in a 3m by 3m space.

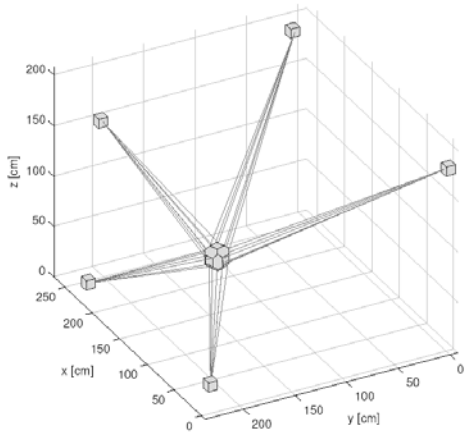


Figure 4.16 Global metrology system time-of-flight distance measurements

Although the actual observability of the system depends on the implementation of the model by each scientist, the SPHERES metrology system allows measurement of all the elements of a standard state vector for dynamics and control for a second order rigid body system such as a SPHERES satellite: position and velocity (linear and angular). Figure 4.17 illustrates how the state vector for each satellite can be filled by the use of the global metrology system and the inertial measurement sensors.

Each satellite also includes its own beacon to accommodate a different type of state vector: differential states between two satellites. This state vector can be used in the development of docking algorithms as well as formation flight maneuvers which require direct measurements between satellites, rather than by finding the differences using the global system. Figure 4.18 illustrates how the metrology system can be used to fill the state vector between two different satellites. The satellite beacon systems have a limitation due to the beacon location (directly on the X axis): it is not possible to use the pathlength differences to determine the attitude between the two satellites along the X axis.

4.3.2.2 Flexible communications: real-time & post-test download

Selection of metrology is not good enough. We need to make sure we can save all the data we need. The team had to recognize the limitation of the wireless communications bandwidth and account for that. Therefore the communication system is flexible in several

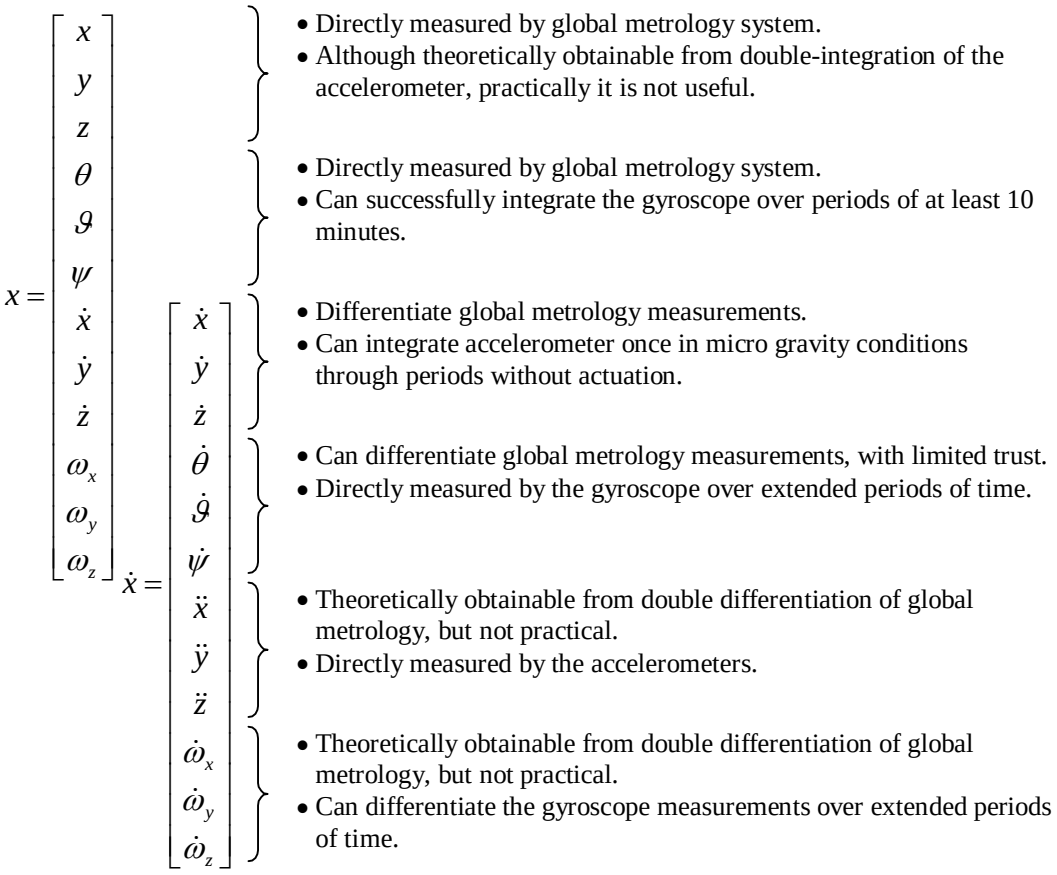


Figure 4.17 Measuring the state vector with the layered metrology system

ways: it has two types of data download (real-time and post-test) and can use custom protocols for inter-satellite communications.

The real-time data download must be transmitted within the TDMA windows of the satellites, meaning that data can be sent between units at up to 5Hz (200ms frames). Each communications packet consists of up to 32 bytes of data and takes approximately 15ms to transmit. The total amount of data that can be transferred in real time ranges from 320 bytes at 5Hz (12.8kbps) for one unit to 64 bytes at 5Hz (2.56kbps) for five units. Since synchronization of the STL channel with the laptop is essential for the test management elements of the SPHERES software, the STL channel must always remain under the TDMA protocol and strictly limited to these data rates.

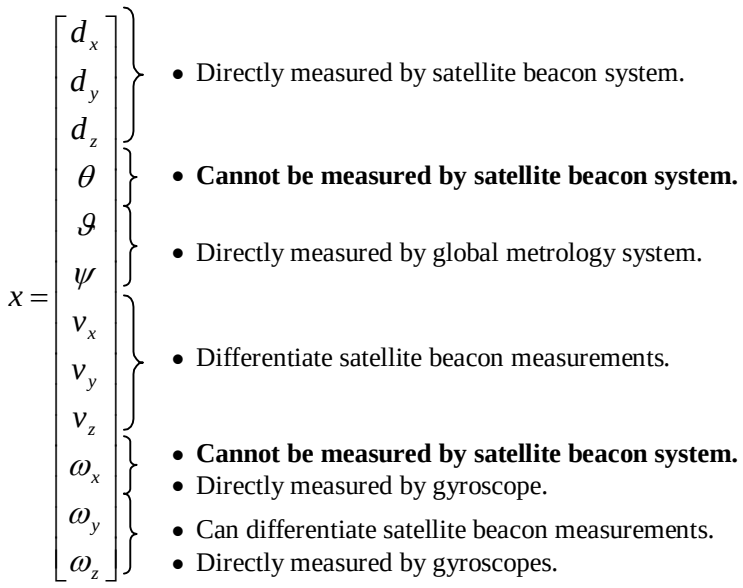


Figure 4.18 Differential measurements between two satellites.

The SPHERES layered metrology system allows full observability of a second order rigid body system in an inertial and a reference frame.
The satellite beacon system further allows limited direct measurement of distances and angles between two satellites.

The communications bandwidth creates a necessity to trade-off data download and operations time. The communications bandwidth is large enough to help scientists see real-time results of debug and processed data in the laboratory environments and to capture processed data in the space environment (since many of the algorithms will have been tested to trust the data). But it can be too limiting for tests which require high frequency data capture or when several units are used. When low-frequency data capture is enough or once processed data provides the necessary information, all data download can occur in real-time. But if large amounts of data must be saved to provide the necessary information, SPHERES allows scientists to download the data after a test concludes.

Of the 16MB of RAM available in each satellite, up to 10MB are available for data manipulation and storage (depending on the specific program). This memory can be allocated both statically or dynamically to create one or more data buffers. A test can be pro-

grammed to continue until all the buffer is emptied, even if the actuation maneuvers have ended. Further, the test management portion of the SPHERES core software can be programmed so that after a test is run the data buffer is left intact and a second test can then be run to download the data buffer; this buffer can be shared among multiple tests and as long as the *download* test is run in between all other tests the data will be safely downloaded. This operating scenario presents a trade-off: utilizing operations time solely to download data rather than to test more algorithms. Scientists will have to balance their need for high-frequency unprocessed data and the available time to operate the satellites. Figure 4.19 presents a sample algorithms where tests 1...N-1 run algorithms and collect large amounts of raw data in two satellites. Tests N and N+1 are run after each of the other tests to download the data. The time spent on tests N and N+1 must be considered as an overhead to the iterative process, but could be essential for the science.

To accomplish this the communications functions interface with the data solely via pointers, rather than by copying data from the original location to a communications buffer. Doing this minimizes the memory space used by the communications interfaces, but it also places the responsibility on the scientists to ensure the data remains safe prior to its transfer. The core software implements four mailboxes of 20 packets; these mailboxes are managed internally by the software to accommodate data of two different priorities in the two communications channels, but are not available as storage space for the user. The SPHERES software routines copy the data from the memory separated by the user to these mailboxes automatically, and transmit the data out of the mailboxes following the TDMA protocol.

The flexibility of the communications system is further enhanced by the ability to use the STS channel using custom protocols and software. While the STL channel is constrained to use the TDMA protocol, the STS channel can be programmed differently and independently. The core software implements a TDMA protocol by default, also operating at 5Hz. The lack of a *ground* station (the laptop) allows inter satellite communications to use the full window to transfer data, with data rates ranging from 2.56kbps per satellite for trans-

```
Test 1
  Init
    Setup comm for multiple units
    Initialize buffer
  Run
    Actuation
    Raw data collection -> Buffer
    Data processing -> RT Comm

Test N
  Init
    Setup comm for single unit (Sat 1)
  Run
    Download comm buffer

Test N+1
  Init
    Setup comm for single unit (Sat 2)
  Run
    Download comm buffer

Test 2
  Init
    Setup comm for multiple units
    Initialize buffer
  Run
    Actuation
    Raw data collection -> Buffer
    Data processing -> RT Comm

Test N
Test N+1

Test 3
Test N
Test N+1
…
```

Figure 4.19 Sample real-time and post-test data telemetry algorithms

mission between five satellites up to 17kbps for one satellite transmitting full time to the other SPHERES.

4.3.2.3 Full data storage

During the initial development of SPHERES strong emphasis was put on the ability to save processed data, ready for real-time display and immediate analysis. The software

> *Flexible communications allow telemetry download in real-time for limited data and post-test for large amounts of data; the use of post-test data download requires the scientists to trade-off the amount of data with operations time.*
> *The inter-satellite communications channel can be programmed independently of the satellite-to-laptop channel to use a wide range of protocols or default to the core TDMA algorithm.*

which operated in the laptop would not only receive the data and check the packet integrity, it would also translate the raw binary data into special structures such as the state of the satellites; the software would only save the processed data. This practice proved to be too restrictive as any change in the communications structures would require not only to modify the software of the satellites, but also the software of the laptop (during the prototyping stages of SPHERES there were at least six version of the laptop software, depending on the data to be saved).

To facilitate data collection of any type the SPHERES communications design defines only a limited number of special packets; further, all data is saved in its raw form as received by the communications hardware. The special SPHERES packets are:

- General Purpose Commands - outgoing: from the laptop to the satellites. These packets send the satellites information on starting and stopping a test, resetting the units and/or control variables, and starts a frame. The packet is transmitted at 5Hz from the laptop.
- Initialization Packet - outgoing: from the laptop to the satellites. These packets transmit the measured setup of the metrology transmitters such that the satellites can use the global metrology system.
- State of Health - incoming: from the satellites to the laptop. The SPHERES core software automatically transmits its state of health at 1Hz. The packet transmits information about the satellite on-time (since the last reset), the current loaded program, tank usage, test management, current operating mode, and the individual satellite's role in a multiple satellite configuration.
- Background Telemetry - incoming: from the satellites to the laptop. By default the core software queues these packets at 10Hz; they are downloaded as the test progresses. These packets transmit the estimated estate of the satellite as determined by either the SPHERES core estimator or a scientist specified function. The frequency of these packets can be modified by the scientist as needed, and can be stopped if desired.

- Debug Packets - incoming: from the satellites to the laptop. A special packet with up to 16 shorts (16 bit integers) which can be used as needed by the scientists.
- "Datacomm" packets - incoming from the satellites to the laptop. These are special packets used to split data longer than 32 bytes automatically. When used between satellites the data is re-assembled automatically on the receiving units; the laptop simply saves the raw packets without processing. The protocol which manges these packets enables scientists to transmit large amounts of data without having to worry about splitting it themselves.

No other special packets were deemed necessary by the SPHERES team after substantial use of the satellites in several environments and development of the interfaces (described below) together with NASA.

Only these special packets are processed in real time in either the satellites or the laptop by the core software. General purpose commands and initialization packets are processed only by the satellites. The state of health packet is processed in real time by both the satellites and the laptop; the satellites use the information to configure multiple-satellite tests. Background telemetry is processed in real-time in the laboratory environment to expedite tests; the satellites also process these packets in case the scientists need to share state information between units. Debug packets are only processed in real-time by the laptop station in the laboratory environment to help test algorithms. Datacomm packets are only processed in real-time by the satellites to allow scientists to share large amounts of data between units in real-time; the datacomm packets are re-assembled post-test from the laptop telemetry.

In all cases, including outgoing laptop packets, all received data is saved by the laptop programs intact. The full received data allows scientists to create their own data types for examination after the tests without restrictions. Further, it allows scientists to perform error detection and even error correction post-test. Scientists can save data manually and even design algorithms which would not be possible to run in real-time to detect lost bytes and save data in case of communications errors.

In all cases the full incoming and outgoing data from the laptop are saved intact. Only a limited number of clearly defined packets are necessary for real-time processing of data, and even these are saved as received.

4.3.2.4 32 bit floating point DSP

The selection of a computer which would allow scientists to perform their calculations with the needed precision had to be balanced with the need to minimize its size, mass, and power consumption (and in turn heat dissipation). The scientific goal of SPHERES (mature formation flight algorithms) strongly called for the use of a processor which would allow a large number of precise mathematical calculations. The majority of COTS micro-computers utilize fixed point MCU's. Several fixed point processors are capable of more than 1 GIPS, but their performance with floating point calculations drops considerably (up to 16 fold); their benefit is a small form factor and small power consumption. Standard 32bit microprocessors used in PC's, although powerful and capable of over 1 GFLOPS, are prohibitive in their power consumption. Still, the SPHERES team decided that a floating point processor was essential for the success of the mission.

The selected 32 bit floating point Digital Signal Processor allows scientists to not only collect high-precision data (at this point the limits on raw data collection lie within the sensors, and not the processor), but to also process this data without loosing any precision from the original measurements. The scientists does not need to worry that utilizing floating point numbers will hinder the ability of the processor to maintain accuracy or have enough processing power. Further, the inherent support of floating point values by the processor makes it trivial for these numbers to be transmitted between units and saved in telemetry without any overhead to the processes.

The use of a 32 bit floating point processor allows scientists to perform precision mathematical calculations to maintain data integrity through any data processing.

4.3.2.5 No precision truth measure

The conclusion on the data collection and validation aspects of the SPHERES facility must conclude with the remark that the facility lacks a precision truth measurement system. Throughout all operations video will be available, sometimes in real-time, to allow scientists to observe the behavior of the units. This video can be post-processed to validate maneuvers in a coarse manner. It could be possible to add markers to the satellites such that image processing software could help calculate the behavior of the units from the video. Still, even with these options, the team has determined that there is no truth measure which can provide data of the same precision as the metrology system of the facility. In the best cases the telemetry from the satellites will be corroborated by coarse motions in the videos.

4.3.2.6 Redundant communications channels

While the selection of 2 communications channels was due to simulating both satellite to ground and satellite to satellite communications, the SPHERES facility allows these two channels could be used interchangeably to enhance the reliability of the testbed. In the case that one of the channels fails, the second channel can substituted with minor changes. The default configuration defines STL as the 868MHz equipment and STS uses 916MHz. Two laptop interfaces, one of each frequency, are always available in the laboratory environment, and two will be sent to the ISS. If an 868MHz channel fails in a satellite or the laptop, it can be replaced with the 916MHz channel. The core software, which interfaces between the hardware and the user program, was designed to allow changing channels with a single command in the initialization routine. Further, the bootloading software, which allows new programs to be loaded into a satellite, interfaces with both communications channels in identical manners. Therefore, in the case of a failure, new software can be programmed which swaps the use of the channels, ensuring the availability of telemetry and continued (even if limited) operations.

> *The ability to use the two communications channels interchangeably increases the reliability of SPHERES.*

4.3.2.7 Test management & synchronization

As described above, the SPHERES core software enables scientists to program *families* of tests at a time, such that running these tests has minimal overhead. The SPHERES test-management software provides several other features to enhance the repeatability and reliability of tests:

Automatically run the initialization code whenever a test starts. The SPHERES core software provides scientists with two initialization areas: program and test. The program initialization code sets up the satellite properties which will be in effect for the full family of tests. These initialization steps only take place once upon boot. To simplify the repeatability of individual tests the code also provide a routine which is run before a test is started, without the need to reboot the satellites. This initialization routine can be used by the scientist to ensure that all initial conditions are set correctly every time a test is started. Further, the independent initialization routine allows scientists to use the same control code to test for different initial conditions without the need to program multiple controllers. These initialization routines are expected to be simple and quick, so that the controller can start immediately after the routine ends. If the scientist requires further initialization, they can programming the first maneuvers of a test to satisfy the initial conditions before the actual test maneuvers are conducted.

While the core software provides the functions for the scientist to initialize their code, it cannot guarantee that an individual scientists will initialize their program correctly. The responsibility to initialize their code correctly still lies within the researchers.

Manage the controller timing independently of the controller itself. A critical part of control algorithms is the correct timing of the controller. The majority of control algorithms use periodic processed to determine the actuation of units. The SPHERES core software accounts for this need by separating the timing of the unit and the controller

function from the control function itself by utilizing software interrupts of different priorities. Two high priority interrupts, driven directly from hardware timers, maintain the timing of the controller and the units time management. A middle priority software interrupt, triggered by the high priority timing software interrupt, performs the control task itself. While this does not guarantee that the controller software will terminate within its allotted period, it ensures the reliability of the timing information provided in the telemetry. This reliable timing information can then be used to identify controllers which overrun their allotted periods.

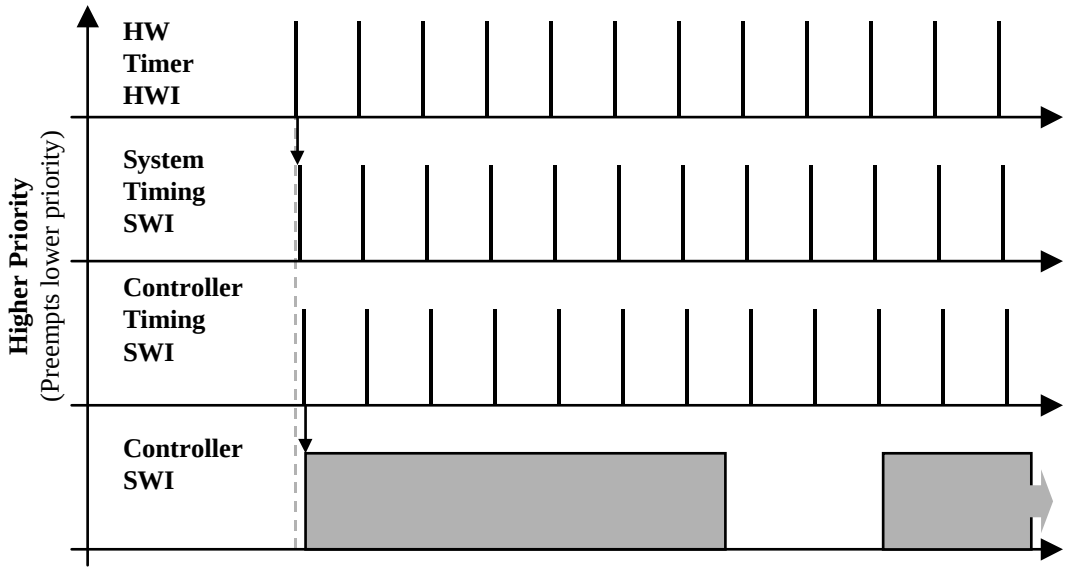


Figure 4.20 High priority scheduling of system timing and controller interrupts

Automatically synchronize test starts among multiple satellites. The ability to repeat tests of multiple separated spacecraft with ease depends on the ability to synchronize all the satellites every time, such that the same initial conditions can be repeated and controlled. Because SPHERES was designed to test algorithms of multiple satellites, the SPHERES core software integrates functions of the communications, avionics, and software sub-systems to synchronize the start of tests to within 1ms.

While the timing of the laptop synchronization packet is not precise, since it fluctuates up to 20ms, its reception by all satellites occurs at exactly the same time. This feature is used by the test-management software to define the time "-1000ms". A general purpose command which includes a "test start" also requires the packet to be acknowledged by all units. The laptop awaits the acknowledgement in a state of health packet, which is created immediately when the start command is received and transmitted in the next available frame after it is created. If the laptop does not receive the acknowledgements of all units within three frames (600ms) it will send a new start command, which once again resets the start time to "-1000ms". If all acknowledgements are received the laptop simply awaits for data, allowing the units to reach test time "0ms", which is the start of the test. Figure 4.20 illustrates this timing sequence for the operation of two SPHERES. In this example satellite number two looses the original start command, and does not send its acknowledgement. Therefore in the third frame after the original start command the laptop sends a new start command packet. Both units receive this new command and after 1000ms start the test synchronized to each other.

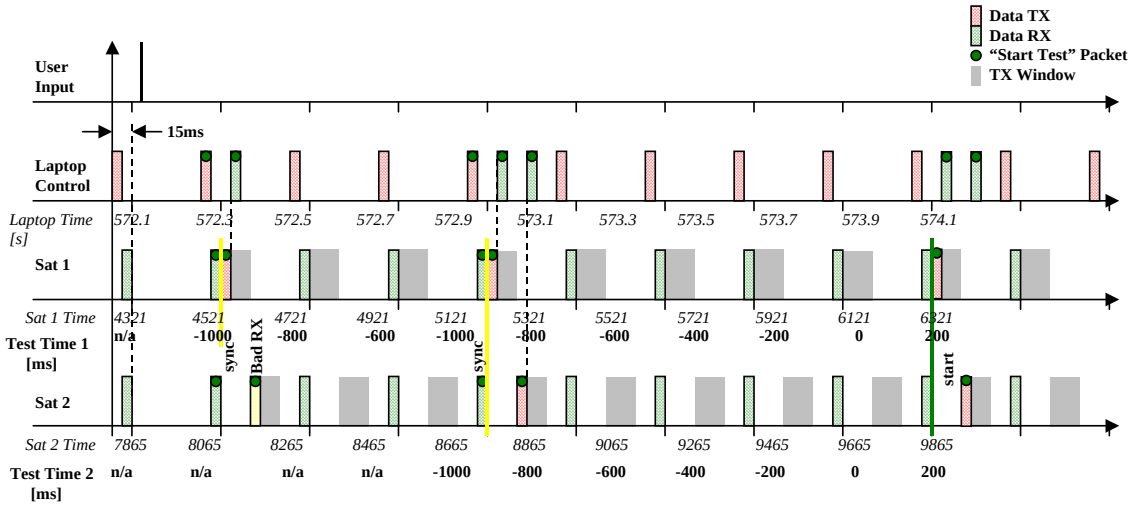


Figure 4.21 Test synchronization via communications.

The synchronization within one ms is achieved by utilizing high priority, non-preemptable hardware interrupts to process the general purpose command packets, while all other

packets are processed with variable latency in a separate lower priority software interrupt. The general command packet is always received by all units at the same time; the transmit time variability is within 60µs. The hardware interrupt has a fixed latency of 80ns and is processed within 60µs. The only variability is due to clock drift between the units. Since the internal time of the SPHERES is maintained to a 1ms precision, the variability of the start time is within 1ms. Figure 4.22 illustrates the fixed time to process a test start command as compared to the general processing of other commands.

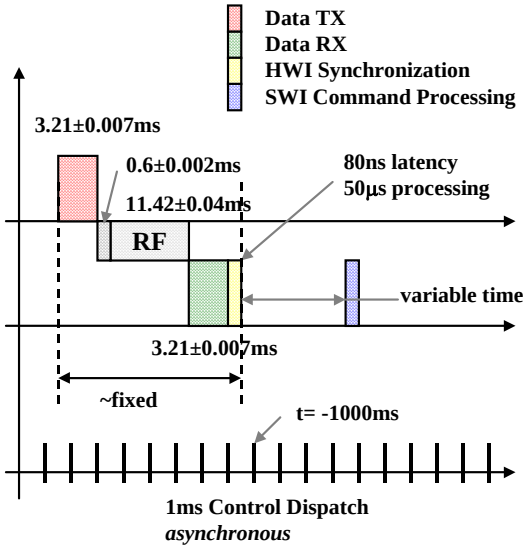


Figure 4.22 Test synchronization to within 1ms

Provide test timing information for post-analysis. Figure 4.21 illustrates a function of the test management software which contributes not only to repeatability and reliability of the tests, but also to good data collection: redundant test start packets. The state of health packet which is created immediately after a start command is received can be used to synchronize all the data between the units, since it provides the exact (to within on 1ms) SPHERES satellite on-time, used in the standard telemetry, at which point the start command was received - all tests start 1000ms after that time. The data management software sends a second start test packet immediately after a test starts, indicating both "test time 0"

and the satellite on-time. The redundant packets ensure that the collected data of multiple satellites can be synchronized.

> *The SPHERES test management procedures initialize tests, control periodic functions, and synchronize the start of multiple units. Redundant data is downloaded to ensure the data can be synchronized after the tests are performed.*

4.3.2.8 Location specific GUI's

While SPHERES clearly depends on humans to manipulate the satellites, the observability by humans is not necessarily guaranteed by the physical nature of the tests. For example, once precision alignment algorithms start to be tested, it may not be possible for humans to determine through without any other help if a test performed better than previous ones. Therefore, the correct instrumentation must be provided in the different environments to allow observability of the experiments by humans, so that they can make the correct evaluations on when to proceed with new tests and when to repeat them. At the same time the user interface play a major role in the ability to quickly repeat tests; the presentation of the correct data must not hinder the ability of the human to observe the test by distracting them with data overload, nor should it prevent the operator from quickly starting new tests.

The ground based GUI is a streamlined interface consisting of one main dialog window. This window provides real-time information on the status of all powered satellites with active communication links. To provide the user with enough feedback, the GUI displays the state of the satellite, its up-time, and current test information (test number, test time, and maneuver number). This information is useful in ground-based facilities since the operators are either the researchers themselves, or MIT SSL personnel with deep knowledge of the tests being conducted. The ground-based GUI also provides direct access to control the satellites with minimal need for input. This includes the ability to start and stop tests with a single keystroke, to force a hardware reset of the units remotely, and to reset

any communications channels. The ground based GUI also provides a streamlined method to upload new programs to the satellites, although this method requires direct knowledge of the individual files that must be loaded (as will be explained, the flight GUI adds elements to require less knowledge of the files, but the process adds more steps). The ground-based GUI also includes optional, separate windows, to show the state and debug vector of each operating satellite. This real-time presentation of data allows the researcher to immediately see if the satellites are calculating their state correctly, and reduces the time spent when a test is not operating correctly. By making these windows optional, this GUI ensures that only the information the researcher desires is present, simplifying the operations if so desired. Figure 4.23 shows a screen-shot of the ground-based GUI together with its optional display of real-time data (state - position, velocity, angle, and angular rate - and debug packet).

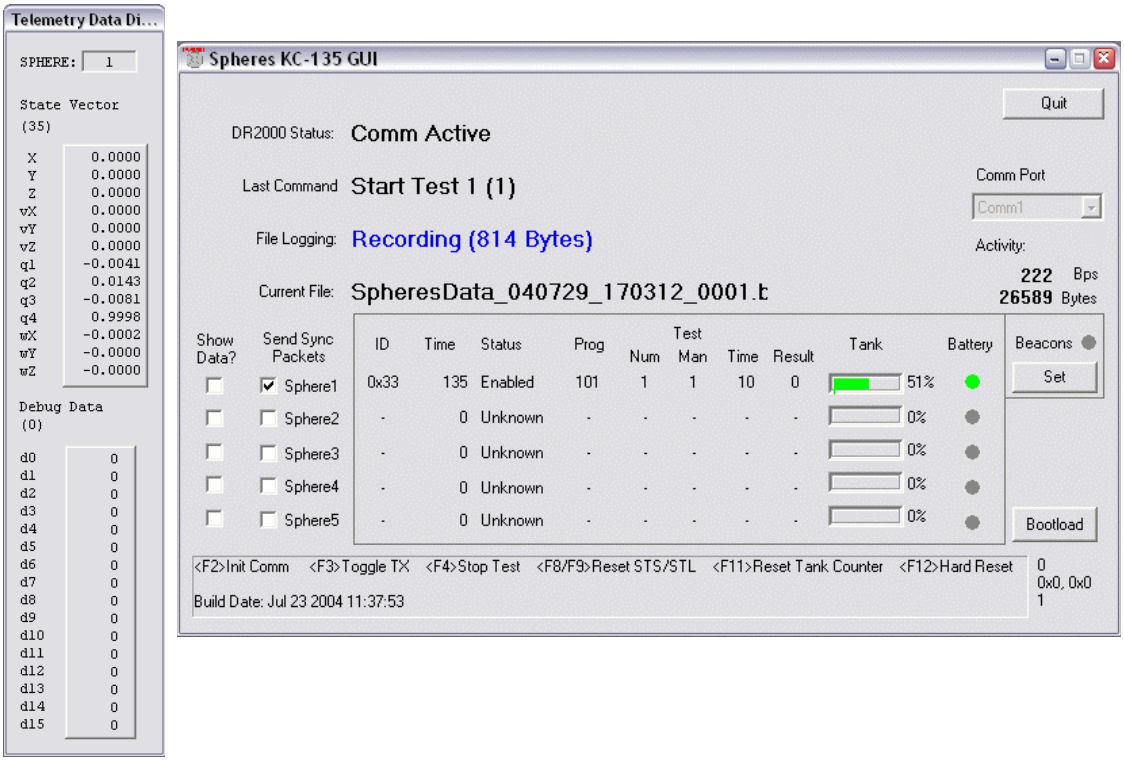


Figure 4.23 SPHERES GUI for ground-based operations

Operations aboard the International Space Station must consider the fact that the operations take place in a remote environment; the researcher is located at their ground facilities, while astronauts operate the tests. While the astronauts will have some knowledge of the SPHERES facility and its operation, they will not have the deep knowledge of neither SPHERES or the science being conducted as the SPHERES team members or the research scientists. Therefore, the SPHERES operational elements within the ISS provide a special graphical user interface which provides the astronauts with enough information to conduct the experiments, without them having to know the system or science deeply. On the other hand, the GUI also provides multiple points of input for the astronauts, to help determine the success of each test run.

Operations within the ISS must meet several NASA requirements that involve both safety factors and interface requirements. The safety factors that affect the operations include the need for the astronaut to require a positive action to enable the satellites prior running tests; the satellites must not perform any thrusting activity prior to being enabled. Further, the satellites may not transmit any communications unless they are in range of the control laptop and the laptop has enabled communications. The interface requirements include the need to maintain a window in the foreground if it provides an action to start or stop a test; the graphical requirements of the interface are not addressed in this section, since they did not cause any changes on the research aspects.

Figure 4.24 shows a screen capture of the flight GUI with the test introduction window displayed, as well as the test start window. The GUI provides the following basic information to the astronaut: status of communications, satellite enabling, battery charge, and tank fill level. The GUI also informs the astronaut of the program currently loaded, and the test (if it is running). Note that the ISS GUI does not inform maneuver numbers, since these are only used by the scientists or SPHERES team members for debugging or data analysis purposes.

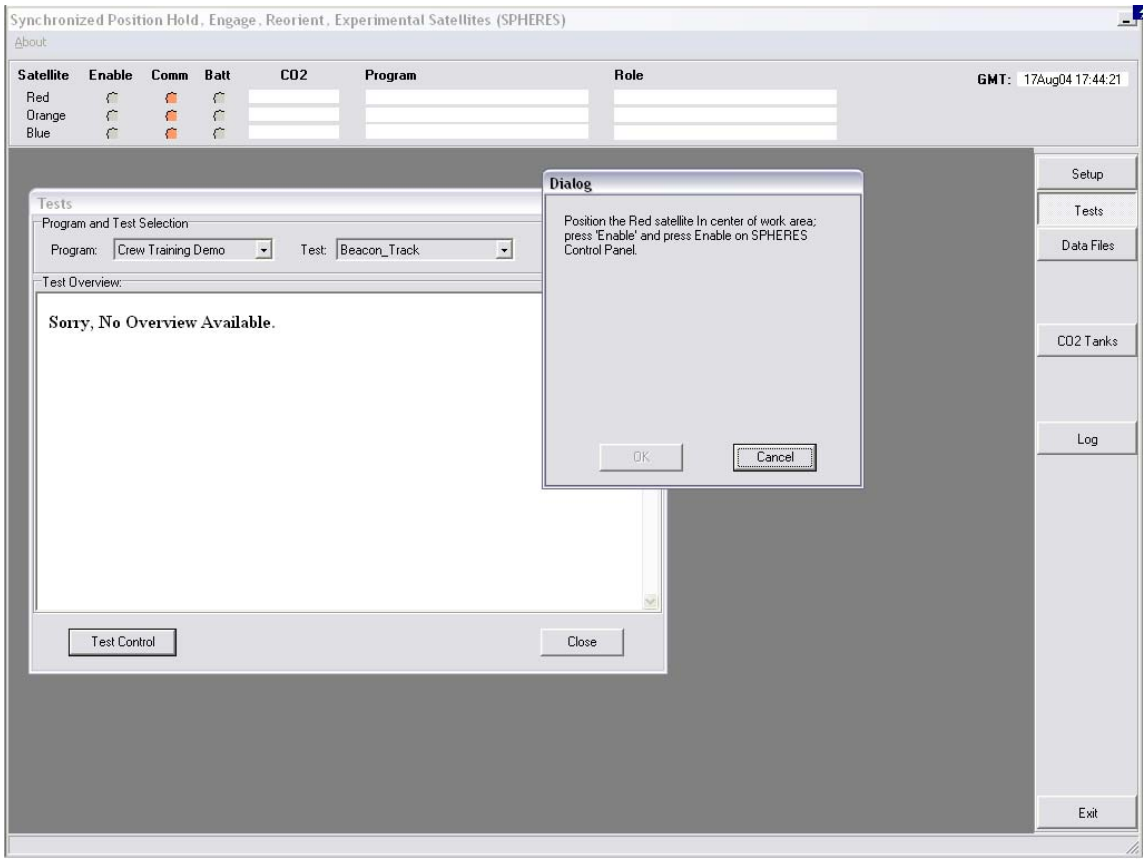


Figure 4.24 ISS astronaut interface

Within the test window the ISS GUI provides a description of the selected test, including the expected behavior, positioning, and in some cases either a picture or a movie that give the astronaut a preview of the test to be performed. This description and preview are especially useful for the iterative research process, since they provide the astronaut enough knowledge to determine on their own the success of a test. Using this knowledge, the astronaut is given the liberty to decide when to repeat tests and when to move on into the next test. While the astronaut will not perform any data analysis, the ability to determine success of a test maximizes the amount of useful data from each time-limited test session aboard the ISS.

The SPHERES software will automatically determine when a test finishes (the astronaut may also cancel the test if they determine it is not proceeding correctly). At that point the

GUI will present the astronaut with a pre-determined termination code (the astronaut will have a look-up table for each program of an ISS session). This code provides the astronaut with further feedback on whether the test was successful or not.

Afterwards, the astronaut is presented with a questionnaire written specifically for each test. The questions are drafted by the scientists to acquire fast knowledge of the success of the test in order to minimize the time spend in data evaluation; if an astronaut provides feedback that a test failed substantially once, but then performed correctly multiple times, the researcher may decide to only look at one good run and at the one bad run, to determine what was different.

Further, the astronaut is given the opportunity to enter free text into the questionnaire form. This open area effectively becomes a lab notebook for the astronauts, where they can inform the SPHERES team and researchers of any problems in executing the tests and any behavior not covered by the pre-defined questions.

Apart from the specialized interface, operations within the ISS will also provide video feedback of all operations. While the astronaut will have substantial opportunities to provide feedback, the researcher on the ground has the most knowledge of the expected behavior; therefore, it is essential for the researcher to corroborate the feedback of the astronaut by looking at both the data and the video of the operations. Past experience shows that astronauts may be more interested in the cases where algorithms do not perform correctly (MACE), rather than successful runs. Therefore, the researcher must be able to determine if a test performed correctly themselves.

> *Two GUIs were designed: one for researchers operating the satellites directly (maximizes real-time data availability and details) and one for astronaut operations (maximizes information on tests, provides summary results, and allows for astronaut feedback).*

4.3.2.9 Re-supply of consumables

SPHERES was designed so that its only physical limitation in mission life are easily replaceable consumables (gas tanks and batteries). Otherwise the design of SPHERES does not limit the mission life, which could be extended for several years. Of course, the operations do require that consumables can be launched to the ISS if they run out, which is not necessarily trivial. Yet, the challenges with sending new supplies are substantially less than with deploying new missions.

SPHERES will operate for at least six months in the ISS, and could be there for multiple years if the research calls for it and the resources permit it. Through this period researchers will be able to conduct extended, iterative investigations. The ability to re-supply consumables supports experiments in three ways:

Simplifies repeatability of tests. Not only does the resupply of consumables allow a large number of tests to occur, it also improves on the repeatability of initial conditions. If necessary, the operations can call for tests that are highly dependent on mass to be performed immediately after the gas tank is replaced, while other tests that do not depend on mass can be performed later. This allows initial conditions to be controlled as necessary.

Because SPHERES was designed to specifically allow the re-supply of consumables, this task was designed to be performed with ease. Replenishing the gas tanks or batteries takes less than one minute, adding only minimum overhead to repeat new tests in case an algorithm failure empties the tanks or batteries run out. Operators, be it in ground laboratories or the ISS, can repeat tests without major worry of consumables as long as replacements are available.

Enables extended investigations. The selected consumables are easily removable, even without being fully depleted, and have extended shell life whether used or new. Therefore, the researchers have wide flexibility in conducting their experiments. The consumables only deplete during the actual operations, and can be stored safely in between tests. In this

manner scientists have the ability to analyze data and modify algorithms over extended periods of time.

Creates a risk-tolerant environment. The ability to re-supply consumables allows researchers to continuously push the limits of their algorithms. Since depleting consumables does not result in the end of the mission, scientists can perform tests which could potentially deplete the propellant, but which could otherwise provide substantial insight into the science behind the algorithms. The ability to replenish the propellant allows scientists to test the high-risk but high-payoff algorithms which cannot be performed in other environments. By allowing researchers to find the true limits of the algorithms, each of their research iterations will be more productive.

> *The ability to resupply consumables provides three major benefits for experiments: provides repeatability, enables extended investigations, and allow scientists to push the limits of their algorithms.*

4.3.2.10 Operations with three satellites

The main driver in the final configuration of three satellites for the SPHERES facility was the need to perform substantial formation flight maneuvers with multiple satellites. This selection has a secondary effect: it improves on the reliability of the testbed. While three satellites will be needed to demonstrate several formation flight algorithms, the use of two satellites can still enable a substantial amount of science for distributed satellite systems.

> *Full understanding of the science needs allows layered reliability of the facility: with SPHERES the deployment of three satellites provides redundancy for tests that require one or two satellites.*

4.3.2.11 Software cannot cause a critical failure

The importance of separating the software from any safety controls was presented above. From the perspective of the NASA safety panel that would be enough; the software could

potentially cause a mission failure, as long as safety is not at risk. From the perspective of the SPHERES design plan the goal goes one step further: the software cannot cause a mission failure. This ensures that scientists can develop their algorithms to their limits; regardless of the program created by the scientists they are assured that if their program fails, they will be able to load new programs to try again.

The design of the SPHERES core software operating environment does not directly control the ability of any other sub-system to perform its functions, only how the data managed by the other sub-systems is processed. The core software could be fully redesigned without causing any failures of the equipment. In other words, the operation of any individual sub-system does not depend in any way on the operating software. The following points describe the de coupling of the software from the other SPHERES hardware subsystems.

- Communications. The communications sub-system interfaces with the core software via both inputs and outputs. The inputs consist of data to be transmitted and configuration commands. The outputs are data received and configuration command confirmations. The communications sub-system operates via two levels of firmware which isolate it from the core software. The processor, which runs the core software, cannot modify that firmware.

  The failure modes which can be caused by the software are purely operational. For example, the software could configure the DR2000 hardware incorrectly, preventing a satellite from communicating. Upon rebooting the satellite, the DR2000 returns to its default configuration. The software can also saturate the micro-controllers which transfer data to or from the DSP, or configure the data transfer rates incorrectly. The firmware automatically discards excess data, ensuring continuos operations; a problem of excess data is corrected automatically once the software reads or writes data at the correct rate. Like the DR2000 hardware, the microcontroller firmware returns to a valid configuration upon reset. The core software cannot cause the communications sub-system to fail permanently.
- Propulsion. The propulsion sub-system interfaces with the software via twelve digital output lines; there are no other interfaces. This sub-system requires special timing on the signals which actuate the solenoids. This timing is performed by external circuitry, which takes as its only input the digital signal, which indicates whether the thruster should be open or closed. The external circuitry determines which signal to create.

The solenoids do have a limit on their actuation frequency (50Hz); the external circuitry does not limit the frequency of operations to within this limit. Therefore it is potentially possible for the core software to drive the solenoids beyond their operational limits to the point of failure. Still, this problem would have to occur for a prolonged period of time without notice for a mission-critical failure to occur. The presence of humans in all tests minimizes the probability that the software can cause permanent damage to the propulsion system, since tests which overdrive the solenoids can be stopped and the satellite can be put into a debug mode which does not perform any actuation. At that point the software can be reprogrammed to prevent mission-critical damage.

- Metrology. The metrology hardware is driven by firmware which operates in an FPGA. It interfaces with the software via the general data bus of the microprocessor; its interface is the most complex of all sub-systems. The core software can configure the metrology system widely; it commands the transmission of infrared signals for global metrology, enables the global metrology sensors individually or collectively, configures the A2D conversion rate, and enables the on-board beacon. But the core software cannot change the actual firmware of the metrology system.

  The core software can cause temporary failures in the configuration of the metrology system, which could potentially saturate the processor and prevent operations. Like with the communications sub-system, the metrology system returns to an operational state upon resetting the satellite, and a debug mode can be entered to prevent further operational problems.

  The metrology firmware protects its hardware directly. The firmware prevents the infrared transmitters from being active for prolonged periods of time, which could cause the infrared LEDs to fail. The firmware also limits the A2D conversion rate to ensure that valid data is always available. The on-board beacon protects itself by only actuating within its established limits.

- Power. The power sub-system interfaces to the core software via digital inputs and outputs. The power sub-system provides the core software with a low battery indicator. The core software must continuously toggle the watchdog data line to prevent a hardware reset. The only failure which could be caused by the core software would be to not toggle the watchdog, which would cause continuous reset of the unit until it is put in debug mode. This continuos reset does not cause critical failure of any sub-systems.

- Software. The only mission critical failure which can be caused by the core software lies within the software sub-system itself. The ability to load a new program is the only mission critical software present in the SPHERES satellites. This software, referred to as the "bootloader", configures the satellites into valid configurations upon boot and allows the satellites to enter the

debug mode necessary to load a new program. This special part of the SPHERES software is treated as firmware, and is not changed when a new program is loaded; new programs are loaded into separate spaces of FLASH memory, and the bootloader ensures that it does not overwrite itself. But it is possible that once a valid program is loaded it could overwrite the bootloader, which would cause a mission-critical failure, since the unit would no longer be able to boot after a reset. Therefore, it is essential for the SPHERES team to ensure that the bootloader is not overwritten. The SPHERES core software provides a special interface to access the FLASH memory which restricts writing only outside the bootloader space. As long as scientists only utilize this interface to the FLASH the bootloader is safe. But since a scientist can modify the FLASH directly, the SPHERES team members must validate any software to ensure the bootloader is not overwritten.

All of the failures which can be caused by the core software to other sub-systems are not mission critical; they are temporary failures which can be corrected by resetting the unit and loading a new program which corrects the problem. The correct use of the core software provided by SPHERES ensures that the software created by the scientists cannot cause a mission failure.

> *The ability to prevent the software from causing a mission-critical failure allows scientists the freedom to push their algorithms to the limits of either the science or the hardware. In this manner SPHERES truly provides a risk-tolerant environment for development of new algorithms.*

4.3.3 Supporting Multiple Investigators

The original goal of SPHERES was to develop a testbed for formation flight and docking. These two subject areas constitute a part of the larger field of Distributed Satellite Systems. The MIT SSL identified the following major topic areas for study within DSS:

- Metrology – Each satellite in a DSS requires knowledge of both its attitude and position as well as that of the other satellites. One must investigate the need for absolute measurements (e.g. a radar pointing towards Earth) versus differential measurements (e.g. docking) and between coarse (e.g. radar) and precise measurements (e.g. interferometry).

- Control – The control fields vary over a large range. High-level architecture determines the type of hierarchy in the system (e.g. leader/follower); an example of an intermediate level is fuel-balancing algorithms; low level control includes rigid body control of each unit.
- Autonomy – One goal of DSS is to minimize human intervention. At a minimum, the main maneuvers of the system should complete autonomously; human intervention should only occur at high levels, such as specifying the current task.
- Artificial Intelligence – AI goes a step beyond autonomy by providing the extra advantages of automatic system reconfiguration and error detection and correction, among others. AI technologies in DSS help further minimize human intervention in the case of a problem or a new mission goal.
- Communications – DSS satellites require communications both to ground (high power) and between the units (low power). Each program must study its optimal communications configuration.
- Human/Machine Interfaces – Given the limited interaction between humans and free-fliers in space, the possible uses and interfaces between satellites and humans must be studied.

The final design of SPHERES contemplates the need for research on these areas. The design takes into account that maturation of these technologies will require the cooperation of multiple scientists. Providing a system that allows multiple scientists to participate in a research program creates a set of requirements that cannot easily be defined as a simple list of qualitative specifications. The requirements are qualitative in nature and of a broad scope. The most important, yet broadest, requirement is to provide as much operational flexibility as possible so as to meet the project goals.

SPHERES implements its operational flexibility through the following features:

- Guest Scientist Program
  - Information Exchange
  - SPHERES Core Software
  - GSP Simulation
  - Standard Science Libraries
- Expansion port
- Portability

- Schedule flexibility

4.3.3.1 Guest Scientist Program

Immediately after the design of the prototype units was complete, the SPHERES Guest Scientist Program came under development to create a true relationship between the MIT SSL and the guest investigators elsewhere. Based on past experiences, the MIT SSL knew that the creation of relationships with multiple scientists to use the same facility required the development of both logistical and operational tools which facilitate the interactions and minimize the physical presence of the scientists with the hardware. The GSP became an integral part of the SPHERES program, making use of both the human and computing resources available. The GSP was a major element in the definition of the scheduling of mission operations (requests to NASA) and the main driver in the design of the software interfaces.

The SPHERES Guest Scientist Program consists of information exchange, special tools (software and simulation), and operations plans. The operational characteristics are introduced in Section 4.3.1.1, which describes how scientists make the best use of the iterative design process through multiple iterative loops. One of the layers includes the development of algorithms in-house by use of a simulation, which can be performed independently by a number of guest scientists. Further, the operations of 2D laboratory tests at the MIT SSL have been designed to support guest scientists in multiple levels. This section describes the information exchange and tools developed to support multiple scientists in further detail.

Information Exchange

The initial communications with a guest investigator include delivering the description of the SPHERES testbed, including extensive numerical data (empirical and theoretical) on the characteristics of the satellites. Scientists receive information on the mass properties of the satellites, sensor characteristics and locations, and the thruster profiles. Through the first years of development, and even in ongoing programs, developing a full system-iden-

tification of the satellites has been an integral part of the Guest Scientist Program. This system ID will allow scientists to fully model the satellites to understand the differences between their intended applications and the SPHERES testing facility.

SPHERES Core Software

The goal in the software design was to create an architecture that was relatively easy to learn and flexible enough to accommodate a wide variety of the sophisticated applications in advanced control, estimation and autonomy. The main challenge during this process was to balance the often contradictory goals of usability and capability. The goal of ease-of-use called for a clear and logical model of software operation, and the automation of tedious or non-productive tasks. In contrast, the goal of versatile functionality suggested an emphasis on real-time performance and a flexible execution model. Clearly, the design must reflect these high level goals within the constraints imposed by the testbed hardware.

The model of structured, user-supplied routines was an attractive framework, and with the processing power available with the flight hardware, a simple operating system could be developed to meet these needs. An operating system was needed to improve interface and execution flexibility, and to allow multiple threads to execute concurrently

The Texas Instruments DSP/BIOS [TI, SPRU423B] real-time operating system, designed for DSPs such as the C6701, is used as the operating system on the SPHERES satellites. This product provides multi-processing capability, inter-process communication, and a number of input/output management tools. This simple OS (or kernel), interacts directly with the hardware and manages many of the details thread and interrupt handling. Through the addition of multiple distinct execution threads, the core housekeeping functions are separated from the test software. This separation ensures that activities such as communications and telemetry processing are not affected by any computationally-intensive algorithms supplied by the guest scientist. In addition, increased flexibility and other benefits of multi-threading are extended to the end user.

Although DSP/BIOS solved of problem of flexibility, it was necessary to take steps to simplify the user's interface to the core software and underlying hardware. Giving the guest scientist general access to the entire OS would give them maximum flexibility, but this approach is undesirable for several reasons. First, to use the DSP/BIOS operating system directly, the user would have to purchase and then learn how to use DSP/BIOS. Second, without knowledge about the structure of the user-supplied code, it would be very difficult for us to guarantee the performance of the housekeeping functions and to meet NASA safety constraints.

As a compromise, the user is provided with a strict framework into which specialized source code may be inserted. Each module is executed when certain conditions are met. This allows the core software to manage the experiment's execution. The user's code does not interact directly with the hardware or with the DSP/BIOS interfaces. This simplifies the guest scientist's learning process, ensures proper operation of critical housekeeping functions, and facilitates the implementation of the SPHERES simulator. The core services also manage communications between the different processes. This helps to prevent race conditions between the periodic and aperiodic processes by ensuring atomic functions are used when required. Critical variables are accessible only via functions that have been designed to guarantee the preservation of data integrity. Although this model is not flexible enough for general-purpose computing, it is well-suited to the specific applications of estimation, control and autonomy for which the SPHERES testbed has been designed.

The SPHERES Core Software (SCS) layer performs two functions. First, it acts as a buffer between the user-provided experiment code and the operating system and hardware. Mediating between these layers, the core services control the execution of the user-configureable processes and encapsulate the operating system and hardware-specific interfaces. Second, this layer performs a number of background activities that are critical to successful operations. These functions are summarized below.

- **Communications.** SCS is responsible for receiving and processing incoming communications packets, and for transmitting out-going messages when allowed to do so by the TDMA protocol. The communications module also manages transmission and reception of the messages generated by the experiment code, such as custom telemetry or command data. If a data transfer is too long for a single packet (32 data bytes), the communications module segments the transmission and sends one packet at a time. The communications module on the receiving sphere automatically reassembles the original message from the constituent packets.
- **Housekeeping and Telemetry.** The SCS performs a number of routine tasks automatically, without direct command by the user. During normal operations, the spacecraft monitors the tank fill status (by tracking thruster firing), battery charge level, and operational mode. In addition, automatic processes perform a rough estimation of the satellite state. These data are broadcast over the "State of Health" packets previously described.
- **Propulsion.** SCS interfaces between the user code and the digital outputs to the propulsion hardware. The simplest operating mode allows the user to command a fixed-duration firing. This approach mimics the standard practice on-board most real spacecraft. SCS also implements pulse-modulation and provides an approximation of continuously-variable control over force and torque.
- **Test Management.** The SCS implements the test management functions described in Section 4.3.2 above. It monitors the crew commands, and then initializes and begins the user's test. Once the test completes, the software disables the user code, the thrusters, and the active sensors. During the test operation this module ensures that the user code is run at the correct time and communication bound for the GSP layer is received correctly.
- **Metrology.** The SCS implements a special thread to run the MIT designed kalman filter routines in the background. Guest Scientists are given access to the data created by this module and the option to run their own metrology algorithms in parallel or in place of this module.

The usefulness of the GSP hinges on the interface to the user's code. The relationship between the SCS modules and the guest scientists interfaces is depicted graphically in Figure 4.25. The next sections describe the SCS execution model, which controls the threads, and the supplemental libraries which provide support for a wide range of sicentists.

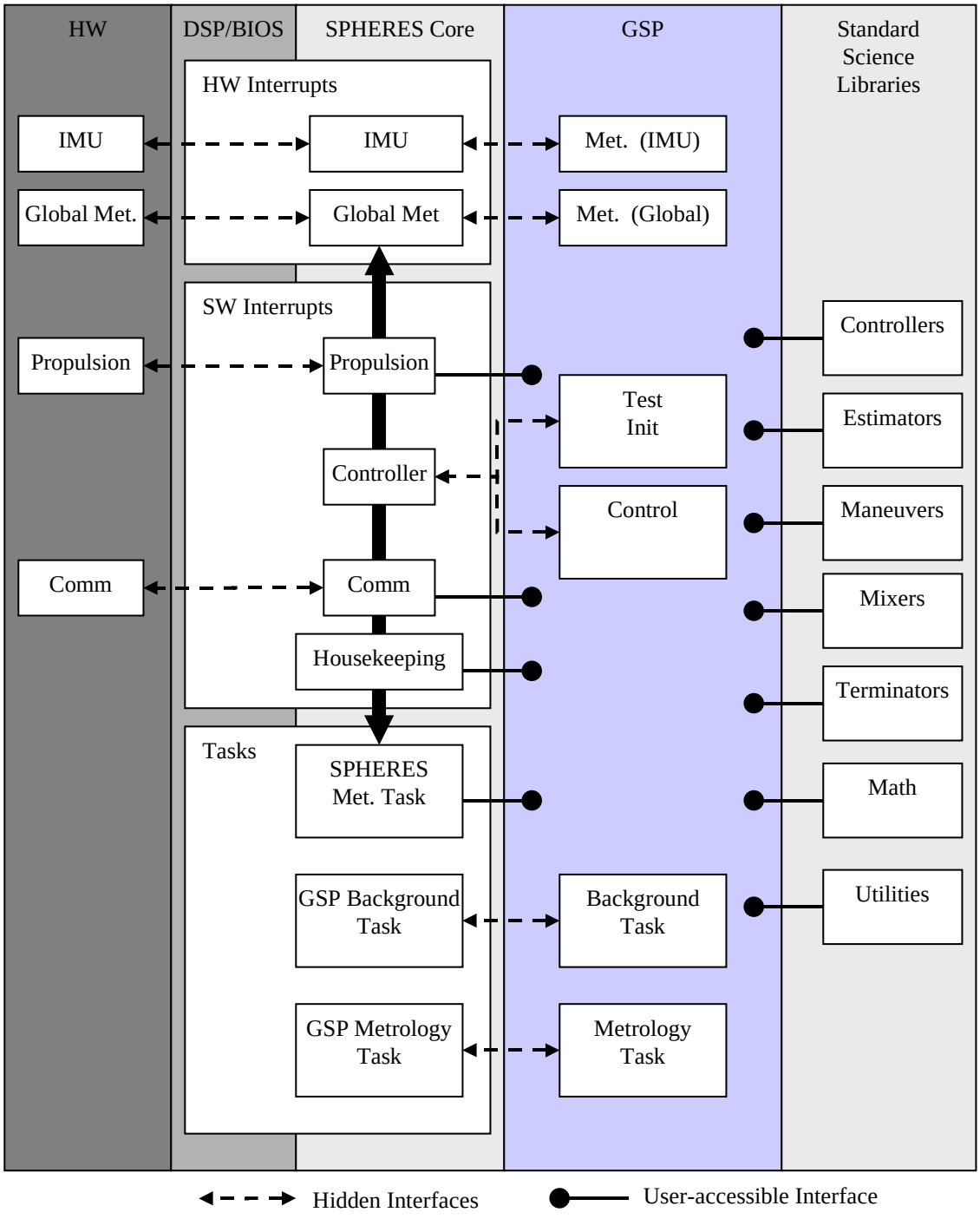


Figure 4.25 SCS interfaces to user code, DSP/BIOS, and hardware

The Execution Model

When writing experimental algorithms for SPHERES it is important to understand the manner in which the code will run. As mentioned earlier, the software framework describes certain modules that the user must provide. These modules are executed by the SCS layer when particular conditions are met. Some modules execute periodically, others in response to events such as incoming communications or sensors.

An important feature of the SCS architecture is that the code is multi-threaded. The highest priority thread waiting to execute is given control of the processor. This helps to guarantee that real-time deadlines are met. Although users cannot create arbitrary threads, they can mix periodic and aperiodic processing.

Guest scientists are provided with the module interfaces presented in Table 4.9 to develop their algorithms. These modules fall within four main threads of the SCS: initialization, control, metrology, and background tasks. The functions of each module are explained below.

TABLE 4.9 SCS guest scientist interface modules

Module	Thread	Repetition	Priority	Time Avail.	Typical Purpose
Program Initialization	Initialization	Once after unit reset	N/A	Long	Initialize the satellite for the full program
Metrology - Inertial	Metrology	Periodic; high frequency	High	Short	Capture inertial; sensor data; integrate data
Metrology - Global	Metrology	Periodic; low frequency	High	Short	Capture global sensor data
Test Initialization	Control	Once at test start	Medium	Short	Initialize individual test
Control	Control	Periodic; mid frequency	Medium	Medium	Periodic controller
Background Task	Background	Aperiodic or long term	Low	Long	Long term processing of data
Metrology Task	Background	Aperiodic or long term	Low	Long	Long term kalman filters

- **Program Initialization.** This module is run once when the SPHERE is turned on or reset. User code in this module can be used to allocate memory or initialize global data-structures.
- **Metrology.** The two metrology modules are used to capture sensor data and place it in an appropriate space for further processing in lower priority modules. Both modules are high priority to minimize the response time, hence maximizing temporal accuracy of the incoming data. As a consequence, there is only a short time available to perform calculations – typically just enough to store the data and perform some basic processing.

  The inertial sensors (the rate gyros and the accelerometers) can be sampled at up to 1000Hz. Simple integrations or filtering can be performed in this module.

  The global module is triggered when data are received by the ultrasonic sensors; it is triggered once each time a metrology beacon signal is received (up to nine times per global metrology request). Every time the module is triggered its data must be saved, as the current data gets overwritten.
- **Test Initialization.** The initialization code described in the Test Management section above (Section 4.3.2.7) runs in the control thread once each time a test starts. Because the module runs within the control thread it must complete within a short time so as not to overrun the configured control period.
- **Control.** The control thread is a fairly common construct. It executes periodically at a user selectable rate. Standard, discrete control laws can be implemented in this module. Although execution rates of up to 1kHz are possible, most experiments to date operate at 1-20Hz. The controller has a medium level of priority. This gives good real-time performance. Significant calculation can be performed inside the controller, but execution must finish before a control-period elapses.
- **Background task.** The background tasks perform general purpose computation in response to specified system events. During initialization, the user's code selects the particular conditions they want to activate the task. Some of these events are unique to the task. For example, the user may make the task responsive to incoming communication. There is also the option to trigger the task from standard actions such as sensor sampling. Once active, the low-priority nature of the task allows long-term background calculations, without the risk of disturbing time-critical periodic activities.
- **Metrology task.** The metrology task allows scientists to perform long term estimations with a direct link to the metrology data, and without the need to program other types of long term estimation in the same thread. Like with the background task, the metrology task will not disturb time-critical periodic threads.

GSP Simulation

An integral part of the GSP is the non-real-time simulation of the SPHERES testbed. The simulation was introduced as one step in the iterative research process using SPHERES, it is further detailed here. The guest scientist begins the custom software development process by writing source code that adheres to the rules described in the GSP interface document [Hilstad, 2003a]. The guest scientist compiles this source code and links it to precompiled SPHERES simulation objects; the resulting program is a simulation client, which represents a single satellite in the simulation environment. The build process is simplified through the use of a compiler configuration file and a standardized directory structure, enabling a client to be built in a single step.

The complete simulation environment consists of one server program and up to five concurrently operating clients. The server contains a graphical user interface for specifying values for simulation and test parameters, as well as for displaying run-time feedback to the user. Simulation parameters include the dynamics environment, the maximum simulation duration, and the test number. Displayed on the GUI are the power status, maneuver number, propellant usage, and communications usage for each satellite. Errors, warnings and informational messages are printed to the Simulation Messages window. The GUI has buttons that open dialog boxes for specifying additional parameters such as the satellite initial state and the locations of the ultrasound beacons. Each client has a message window and a single button that functions equivalently to the power button on the satellite. The SPHERES simulation server and three client programs are shown in Figure 4.26.

The simulation supports all aspects of single and multi-satellite SPHERES operations, including start-up and initialization, STL and STS communications, and vehicle maneuvering. The simulation code base consists of almost all of the SPHERES core code, supplemented by additional code that simulates dynamics, communications, and hardware-level interaction. The simulation records the true state of each vehicle at 10 Hz, and saves all STL telemetry as it would be recorded by the laptop control station in the laboratory. A

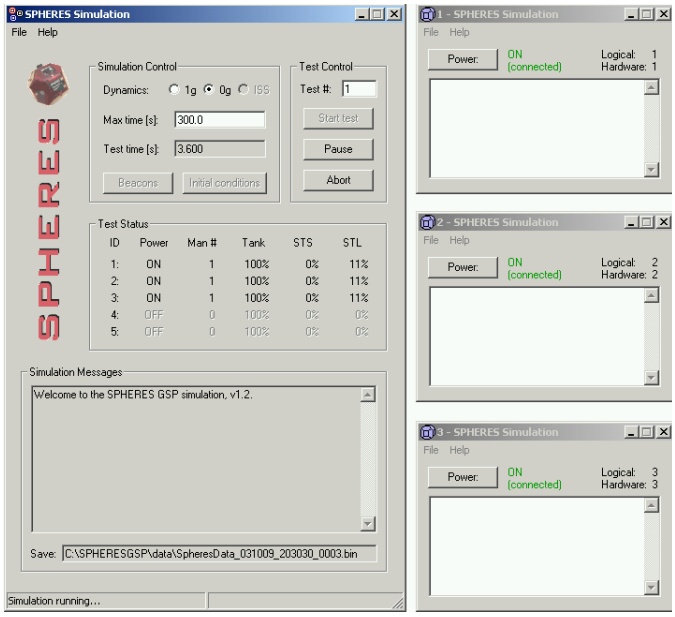


Figure 4.26 GSP simulation window

MATLAB function is provided to read, sort, and plot the data. The simulation is used both to verify syntactic correctness of custom code and to predict the behavior of the hardware in the laboratory and on-board the ISS. Once the simulation has shown that the custom code produces the desired behavior, the code is sent to MIT for verification on the SPHERES hardware in the laboratory.

The simulation guarantees synchronization between the client programs for all timed processes to within one simulated millisecond, the period of the fastest periodic interrupt on the SPHERES hardware. The server enforces synchronization by waiting for all clients to complete each one-millisecond time step before allowing any client to continue to the next time step. This step-by-step process is managed by the server, which sends out step commands and waits for a step completion report from each client. Included in the step command and completion messages are additional data such as state information and communication packets. The clients are multi-threaded, with the main thread handling the user interface and all timed processes, and one child thread running each of five task processes. This multi-threaded implementation allows the use of unmodified SPHERES

source code in the task processes, including functions containing infinite loops, and preserves the free-running nature of the tasks with respect to the timed processes.

Standard Science Libraries

One of the objectives in the design of the GSP interfaces is to minimize the effort that the Guest Scientists must expend on non-productive tasks. For example, if they are interested in developing new estimators, we want to minimize the effort spent on getting the control-system to operate satisfactorily. To this end, we have developed a number of specific function libraries to help accelerate the development process.

The SPHERES core software creates the essential framework to support multiple scientists in the development and maturation of new algorithms. Figure 4.25 on page 170 illustrates the framework created by the SCS on the three left panels; the right-most panel, supplemental libraries, presents an enhancement to this framework which further simplifies the use of SPHERES by multiple scientists. These supplemental libraries are not required by the general SCS framework; they are not operationally required elements. Yet, they transform the SCS API into more than a framework, they create a software platform for the development of DSS algorithms. Through the standard science libraries the SPHERES core software becomes a fully functional facility with basic estimation and control. Individual scientists then take the base SCS environment and create derivative algorithms based on their individual needs.

The standard libraries are optional complementary functions to the SCS. Scientists can select to use the provided functions, provide their own developed independently, or use the standard libraries as a starting point for custom functions. These libraries help provide scientists with guidelines on the development of their own algorithm, but by being optional and independent of the SCS, do not constraint the scientists in any manner.

Figure 4.25 groups the standard science libraries into their major elements:

- **Math**. The library of math functions was developed to ensure compatibility of complex mathematical functions with the C6701 DSP. These functions include standard matrix manipulation routines, inversion methods, and LTI filters commonly used in control and estimation algorithms.
- **Control.** This library includes a number of 1DOF, 3DOF, and 6DOF proportional (integral and derivative) closed-loop controllers. A non-linear switchline controller is also available. These controllers have not been optimized for any specific condition; rather, they have been designed to guarantee stable operations. In this manner scientists who concentrate on other topics, such as estimation or autonomy, need not worry about the development of controllers.
- **Estimation.** The estimation libraries include several different Kalman filter routines. These estimators use both the inertial information and the global metrology sub-system to determine the full state of a satellite; they also include estimators to calculate differential states using the on-board beacons. The standard SCS estimator, which operate in the SPHERES Metrology Task (Figure 4.25), is part of this library. The library also includes other estimators under development at the MIT SSL.
- **Maneuvers.** A range of individual maneuvers, such as single-axis translation or rotation are available in this library. These maneuvers can be combined with standard or custom control and estimation functions to complete a test. The library also includes a set of *terminators*, functions which test when a maneuver and/or test has completed and indicates the fact to the higher level SCS components.
- **Mixers.** The SPHERES GSP uses a broad range of knowledge on the satellites' physical characteristics to provide scientists with accurate mixers which translate a force/torque command into thruster on-off times. These mixers take into account the mass properties, the thruster locations, and the thruster IDs.
- **Utilities.** The standard science libraries provide several utilities not directly related with algorithm development, but which support their development. These include data compression functions for post-test analysis and communications debugging routines for ground-based tests.

Although the libraries are designed specifically to operate in the SPHERES environment, these routines do not issue commands directly to the hardware interfaces. Instead, they perform the requested calculations and prepare a command. Since the user must issue the thruster command there is never confusion or contention about where the command originated, and the scientist always has access to that information.

The Guest Scientist Program is an integral part of SPHERES which combines operational and software features to support multiple scientists. It provides a simulation for inhouse software development. A flexible yet robust software environment creates the execution framework for the satellites. A set of optional standard science libraries creates a software platform upon which scientists can develop their own derivative algorithms.

4.3.3.2 Expansion port

The SPHERES team realized that custom software had realistic limitations in the ability to mature science completely. Maturation with respect to TRL's requires the demonstration of algorithms in representative environments, and the SPHERES hardware could only represent general spacecraft. To ensure that SPHERES provides the opportunity to mature algorithms through higher TRL levels, SPHERES provides for the expandability of hardware components so that the generic SPHERES satellites can be customized with mission-specific science-type payloads.

Each SPHERES satellite has two flat panels on opposite sides that can be used to expand the hardware payload. One side provides a passive mechanical attachment point, where expansion items that do not need any connections to the satellite electronics can be attached. For example, this panel can be replaced by a passive "docking pin."

The other side provides both mechanical and electronic connection points. This side, called the SPHERES Expansion Port (Figure 4.27), interfaces to the main electronics stack via both serial and parallel lines, and provides power for external components. Expansion items can interface to the main processor, allowing all algorithms to reside within the main SPHERES software. The Expansion Port can be used for items such as an active docking mechanism with sensors and actuators.

The design of the expansion port contemplates two needs: easy of integration of simple payloads and the capability to support complex payloads. The port provides three output voltages (+5V, +15V, and -15V) to support standard electronics as well as analog components. Simple payloads are supported via a standard UART serial line (up to 1.25Mbps).

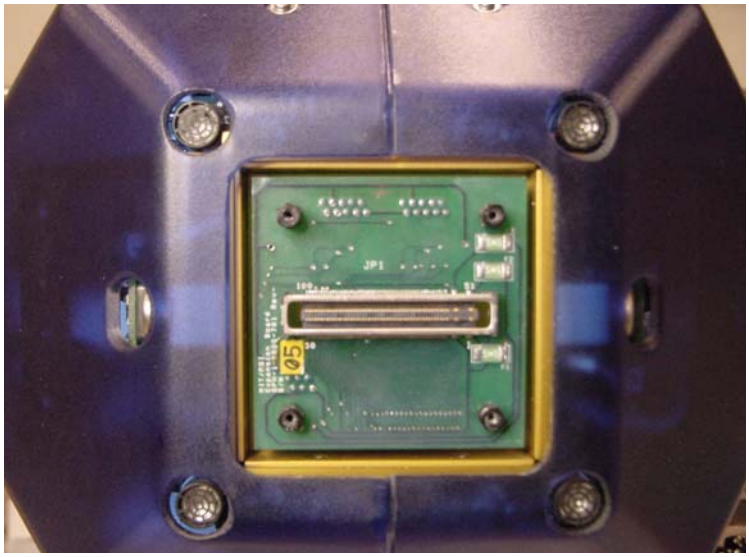


Figure 4.27 SPHERES satellite expansion port face (without cover)

Complex payload communicate with the DSP directly over the processors global data bus, a 2GB 32-bit memory space. Three analog input lines are available directly on the expansion port connector. The expansion board also includes hardware to allow the substitution of the global metrology sensors in that satellite face with new sensors in the expansion item, to account for the case when the expansion item covers the sensors and the global metrology system must be used. A schematic overview of the expansion port is presented in Figure 4.28.

> *The SPHERES expansion port allows hardware expandability for new science payloads. The port provides simple interfaces for quick integration and high capacity memory interfaces for complex payloads.*

4.3.3.3 Portability

A side benefit of the requirement to design the satellites such that they fit within one MLE was the easy of portability of the hardware. Flight-identical hardware can be transported without special considerations. All the necessary hardware for full operations in ground-based facilities can be transported using two to six hard-shell transport cases (depending on the number of satellites to be used), with mass ranging from 30kg to 150kg. Demon-

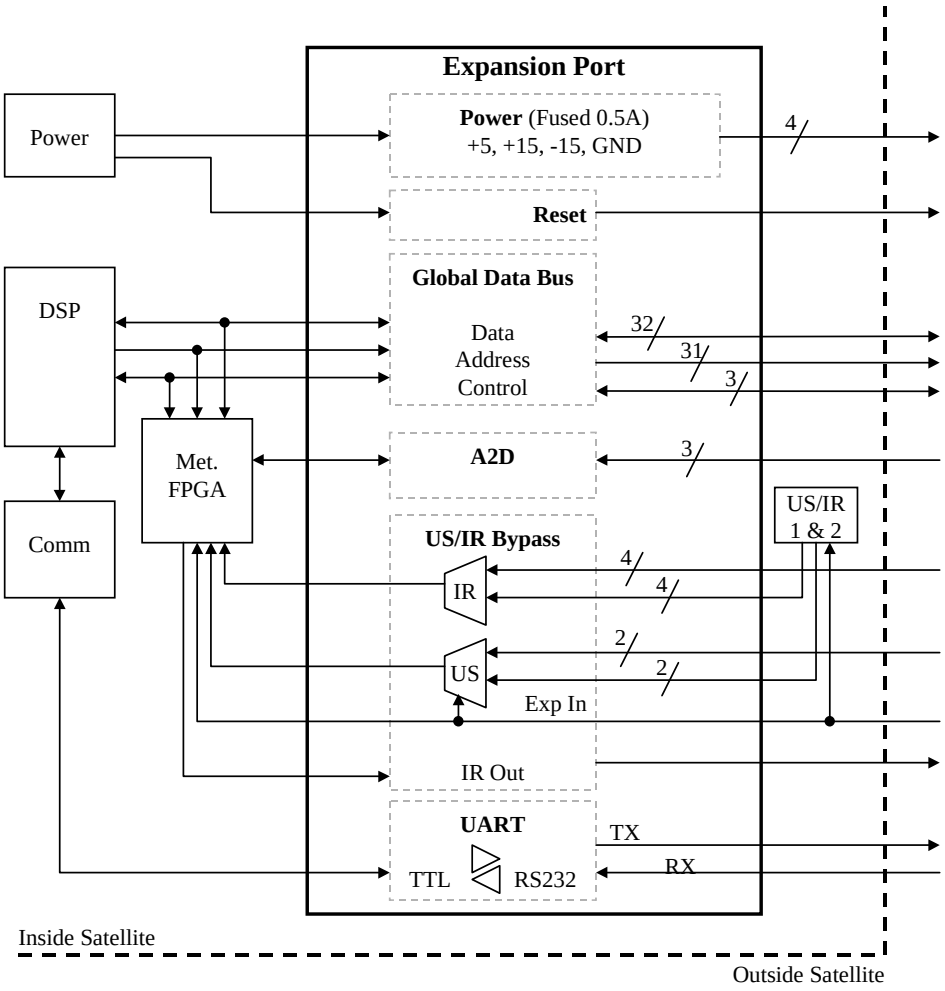


Figure 4.28 SPHERES expansion port design overview

stration of algorithms which do not require science iterations can be performed with only the satellite(s), batteries, tanks, laptop, and a communications box; these can fit in a single hard-shell box at around 20kg.

While not necessarily viable for all types of space maturation experiments, this portability helps SPHERES support multiple scientists by allowing operations in the necessary environments to advance their science. Portability does not necessarily simplify the involvement of multiple scientists directly, rather, it opens the operational environments of SPHERES to support a wider range of environment that become representative of those needed to mature DSS algorithms. The portability opens the operational environments to

locations beyond the MIT SSL and the ISS to other facilities presented in Chapter 1. For example, the hardware can be easily transported to the NASA Reduced Gravity Office for test in the reduced gravity airplane. These tests allow microgravity experiments in a ground-based facility, providing scientists with data beyond that capable at the MIT SSL. The hardware can also be transported to flat floor facilities, when scientists require larger operational areas than those allowed at the MIT SSL or even the ISS. Lastly, the hardware could be sent in a temporary basis to the locations of the scientists themselves.

> *The portability of SPHERES allows the facility to operate in a wide range of locations to better resemble the representative environments required for technology maturation of the different DSS science fields.*

4.3.3.4 Schedule flexibility

From its conception the SPHERES operational plans called for flexibility in the scheduling of operating sessions in the ISS. The baseline plan of one operating session every two weeks drives the frequency of total operations, but does not necessarily constraint scientists to follow that timeline strictly. Instead, the program calls for the MIT SSL to manage the schedule among participating scientists to make full use of each operating session but also to allow scientists to set their own schedule as necessary. The schedule allows the intercalation of scientists so that each session can concentrate on a limited number of science goals (simplifying the work of operators) and allow each group of scientists enough time to review their data between their sessions. At the same time, if scientists only require a small amount of operational time and prefer quick turn-around of tests, the schedule (and core software) allows for multiple types of science to be conducted in the same session every two weeks.

> *Both SPHERES and guests scientists make use of schedule flexibility by ensuring that ISS operating sessions are used in full and that scientists conduct operations as frequently as they need without strict limitations beyond the minimum two week cycle.*

4.3.4 Reconfiguration and Modularity

Modularity formed an integral part of the SPHERES design from its initial stages. The prototype development teams were divided into teams which designed individual sub-systems in a modular fashion: each sub-system minimized its dependence on the others for operations. The SPHERES satellite design is modular. The design of the individual sub-systems can be (and has been) easily integrated into other project which use different configurations due to their simple interfaces and operational independence. Still, once the flight hardware design was finalized and the satellites were assembled, this modularity is no longer visible to the scientists which operate the facility.

The modularity of SPHERES which matters to the scientific community is that which enables wide flexibility in the use of the facility. Through system-wide features and specific sub-system design choices, the SPHERES facilities can be changed to better reflect the science needs of individual scientists. The facility allows for reconfiguration of both software and hardware, as well as flexibility in the use of one or more satellites to create representative environments.

The primary characteristics of the SPHERES facility which enable reconfiguration and modularity are:

- Generic satellite bus
- Science specific equipment: on-board beacon and docking face
- Generic Operating System
- Physical Simulation of Space Environment
  - Operation with three units
  - Operation in 6DOF
  - Two communications channels
- Software interface to sensors and actuators
- Hardware expansion capabilities
- FLASH memory and bootloader

4.3.4.1 Satellite bus

The SPHERES satellites provide generic equipment for space technology maturation experiments by implementing a general spacecraft bus for use by scientists. The primary functions of a spacecraft bus are to support the payload, provide maintenance of orbit and pointing of the payload correctly, and provide power, communications, and data storage. To accomplish these goals, spacecraft payloads utilize the following main sub-systems: propulsion, attitude determination and control, communications, command and data handling, thermal, power, and structures sub-systems [Larson, 1992]. The SPHERES satellites provide each of these sub-systems (except thermal, which is not required in the ISS) and allow the scientist to utilize them in their science as needed. By including all parts of a generic satellite, the SPHERES satellites provide scientists with a true physical representation of an operational spacecraft. Developing a full satellite bus fulfills the need for a physical end-to-end simulation of a spacecraft with realistic physical responses and interactions between sub-systems.

The basic SPHERES satellites enable scientists to mature DSS algorithms for coarse control of systems; i.e., the default configuration, without science-specific expansion items, allows scientists to test algorithms that would perform general maneuvers to initiate and maintain formations, docking tasks, or similar. High precision control can be tested in the future by the addition of science-specific payloads. The SPHERES basic satellite bus configuration provides generic space sub-systems (Table 4.10).

The position and attitude determination and control sub-systems (propulsion and metrology) provide basic actuation and sensors similar to those found on current spacecraft. Actuation is provided by on-off thrusters, providing similar response curves to standard space thrusters. Precision actuators are not provided in the basic satellites: reaction wheels, active optical elements, and other actuators can be added via the expansion port. The metrology system resembles a GPS system, in a local fashion. It provides state information to sub-centimeter precision. This precision is valid for coarse control of spacecraft,

TABLE 4.10 SPHERES implementation of a spacecraft bus

[Larson, 1992]	SPHERES	Generic	Specific
Propulsion	Propulsion	✓	
Attitude Determination and Control	Propulsion and Metrology	✓	
Communications	Communications	✓	
Command and Data Handling	SPHERES Core Software		✓
Thermal	*n/a*		
Power	Power	✓	
Structures	Structures	✓	

but higher precision sensors will need to be added to demonstrate technologies for optical imaging via separated spacecraft.

The communications sub-system selection was based on the need to provide wireless communications, but not driven by the requirements of specific mission. The selection simplified the integration into the ISS. The implemented protocol answers to the behavior of the selected hardware, rather to a specified protocol for DSS. The system allows the protocol to change between satellites, such that the only true constraints are the half-duplex nature of the wireless system (which affects all wireless systems, including existing space communications) and its determined maximum data rate.

The power and structures sub-systems simply ensure the functionality of the satellites. Their design does not answer to any mission specific requirements, but rather to the general need to ensure operations in the ISS and other facilities over extended periods of time.

The command and data handling sub-system (the SPHERES core software) is potentially the only non-generic system; its does not necessarily mimic space systems entirely. Its design was driven directly by the objective to mature test control, estimation, and autonomy algorithms; therefore, rather than simply being a command-handling engine, it also provides routines to specifically meet that objective. At some level it was required that

part of the SPHERES sub-systems specialize in meeting the mission requirements; the software sub-system deviates from the generic nature of the other sub-systems to fulfill these requirements.

> *Each SPHERES satellites is a physical end-to-end simulation of a spacecraft bus. The individual sub-systems are generic in nature, except for the software sub-system which is specialized to meet the mission objectives.*

4.3.4.2 Science specific equipment: on-board beacon and docking face

The initial deployment of SPHERES was driven directly by two specific DSS fields: formation flight and rendezvous/docking. The generic satellite bus provides the necessary tools for formation flight tests; rendezvous/docking algorithms required the addition of science-specific equipment to truly meet the requirements for a physical simulation of the intended systems. To better model docking applications, the SPHERES satellites include two elements specifically designed to enable testing docking algorithms:

- **On-board beacon**. The on-board beacon is a replica of a global metrology transmitter box placed internally on the -X face of the satellites. The design is almost identical to the external beacons, except that it uses the satellite's exiting power sources and infrared receivers, to avoid redundancy in electronics. Otherwise, the on-board beacon includes its own microcontroller and ultrasonic driving circuity, and behaves identically to the external beacons. This beacon interfaces with the satellite avionics so that it can be enabled during tests that require its use and otherwise disabled to minimize power consumption when it is not needed. The internal avionics can configure the beacon number so that it can be used as a stand-alone global system (to determine differential states between satellites, see Section 4.3.2.1), or as part of the larger global reference.
- **Docking face.** Also place on the -X face of the satellites, the docking face is a simple docking mechanism so that satellites remain joined after a docking maneuvers, rather than produce an elastic collision and separate after impact. The docking face consists of a special pattern of velcro strips; the pattern maximizes the amount of angle (roll between units) error allowable so that capture still occurs. The velcro is located around the on-board beacon so that two units can approach each other with direct measurements between them during the docking maneuver.

The -X face "docking face" of the satellites is pictured in Figure 4.29. The on-board beacon ultrasound transmitter is visible in the center of the face. The velcro pattern is shown around the ultrasound transmitter.

> *The SPHERES "Docking Face" provides an example of specific equipment developed to satisfy a specific mission objective: demonstration of docking and rendezvous algorithms.*

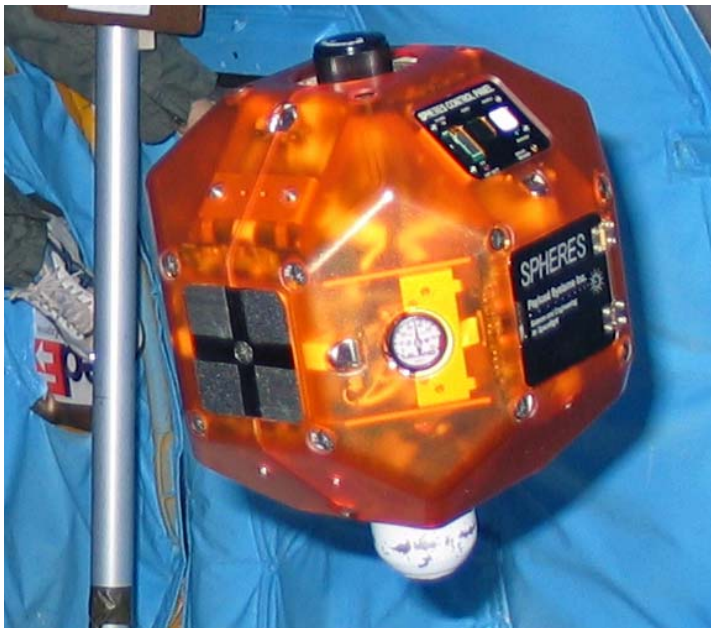


Figure 4.29 SPHERES -X "docking face"

4.3.4.3 Generic Operating System

The software of the satellites must allow multiple researchers to use the general bus provided by the hardware and to interface with any specific equipment added by the scientists. To this purpose, the testbed's software design was almost entirely driven by the need to accommodate multiple researchers. The goal in the software design was to create an architecture flexible enough to accommodate a wide variety of the sophisticated applications within the main areas of DSS. The resulting SPHERES Core Software (described in Section 4.3.3.1) creates a generic operating system for the SPHERES program.

SCS grows upon a real-time operating system (DSP/BIOS) to create a structured framework to develop a wide range of programs using a standard programming language. The development of the SCS in standard ANSI C, with support for C++, generalizes the nature of the operating system. Its use of a generic programming language ensures that a wide range of scientists can develop their algorithms for use on SPHERES. While a custom API was created to the SCS, all of the interfaces are fully complaint with the language standards. DSP/BIOS features generic tools of any RTOS, such as hardware and software interrupt management, pipes, mailboxes, and semaphores. Although scientists do not need to interface with those tools directly, the resulting SCS is based directly on these generic tools; further, scientists can access these tools if necessary.

> *Rather than implementing a generic version of a spacecraft command and data handling program, the SCS implements a generic real-time operating system framework for algorithm development.*

4.3.4.4 Physical Simulation of Space Environment

SPHERES simulates the expected operational environments of formation flight, docking, and other DSS missions closely. To meet the feature of physical-end-to-end simulation SPHERES operates with three satellites in a 6DOF environment using two separate communications channels.

Operation with three units

An important part in the original design process of SPHERES was the determination of the number of units to operate with. Because the primary science goals of SPHERES at the time considered formation flight and docking algorithms, it was clear that an absolute minimum was two units. Two units allows full demonstration of docking algorithms. Two units also allows demonstration of multiple formation flight algorithms, including initial development of any type of algorithms. But the use of two units did not truly meet the feature of a physical end-to-end simulation with realistic simulation of the expected opera-

tional environment for formation flight missions. Intended missions at the time (e.g., TPF and Orbital Express) utilized more than two units in all of their expected operational environments. For example, the use of two units does not simulate the results of two followers maintaining formation with a leader spacecraft, but independently of each other. This simple example results in the requirement to operate three units to fulfill the need for a physical end-to-end simulation.

The use of three units increases the trust on the formation flight demonstrations performed with SPHERES. First, formations can be defined in terms of planes rather than lines; maintaining the plane is essential for imaging applications, and two units could not demonstrate that capability with confidence. Further, the use of three units allows the demonstration of how different architectures [Saenz-Otero, 2000] compare with each other under realistic operations. Leader/follower architectures can operate with multiple followers and show their advantages over master/slave architectures where the slaves are completely blind from each other; peer-to-peer architectures can demonstrate failures in one unit and recovery by the other units. Three units can potentially demonstrate the capabilities of hierarchical structures by defining each of the units as one level under the other. While three units do not model all formation flight missions identically, the use of three units captures the most important physical characteristics which must be demonstrated to mature the algorithms.

Operations in 6DOF

Even in the theoretically ideal case where a physical system has a diagonal inertia matrix and all sensors and actuators are de-coupled along each major axis, expanding the dynamic equations from 1DOF to 3DOF and then to 6DOF is not a trivial process. Adding rotational degrees of freedom adds substantial complexity to all dynamics equations; moving from a 3DOF to a 6DOF system adds two rotational degrees of freedom. The physically realistic scenarios of a non-diagonal inertia matrix further complicates the expansion of problems to 6DOF.

Therefore, to demonstrate algorithms for spacecraft the environment should allow natural asymptotic dynamics to emerge and system dynamics to develop in 6DOF. In order to properly model the system, the full complement of six degrees of freedom are required. In this manner the environment allows traceability and modeling of formation flying maneuvers, especially large out-of-plane coordinated movements.

Two communications channels

The primary driver in the selection of two independent communications channels was to simulate the communications methods of separated spacecraft systems as close as possible. Each of the channels simulates the two types of expected communications present in DSS operations: satellite-to-ground (STG) and satellite-to-satellite (STS). Actual systems will use different systems for each type of communications. STG channels are expected to be high-power, high latency (long distances) systems which download science data to ground after the satellites capture and process information (the STG channels are not necessarily low-bandwidth, since the throughput can be high, but the latency is prohibitive for controls). STS channels are expected to be low-power, low latency, high bandwidth (short distances) systems which transfer data between the satellites necessary to maintain precision formations or perform autonomous docking. SPHERES implements two channels which are operationally identical; their only difference is in the actual RF frequency (868.5MHz vs. 916.5MHz).

The implementation with identical channels helps SPHERES fulfill other features of the MIT SSL Laboratory Design Philosophy, but does not hinder its ability to provide a physical end-to-end simulation. Because the hardware of the two channels is operationally identical and the frequency choice can be easily swapped in software, the only physical limitation of the implemented system is in the available data transfer rate of up to 16kbps for a single unit using the implemented TDMA protocol. Otherwise, the communications channels can be used by scientists with software filters to simulate the different types of communications. For example, if a scientist wishes to simulate a system where only the

master satellite has an STG channel but slave units do not, the software in the simulated slave units can be programmed to ignore all STG communications. Similarly, software filter can implement delays in the STG channel, or limit the throughput of either.

> *SPHERES creates a realistic physical end-to-end simulation of expected formation flight missions by operating with three satellites in a 6DOF environment. The two independent communications channels add further realism to the simulated operations.*

4.3.4.5 Software interface to sensors and actuators

Section 4.3.3.1 described how the SPHERES Core Software mediates interactions between the scientist user code and the DSP/BIOS and hardware. This layer not only simplifies the interfaces to the hardware, it also allows the creation of custom interfaces to the sensors and actuators. Scientists can create a third layer of interfaces to the sensors and actuators which better model their intended operational environments. Specifically, scientists can create filters or special models to interact with the propulsion and metrology subsystems.

The default core software implements standard pulse width modulation actuation via the thrusters. The thrusters are commanded on-off periods of actuation; the basic software immediately implements the commands. Scientists can create functions which first center the pulses on specific frequencies, or they could implement frequency modulation actuation, rather than pulse width. These filters could also add delays in the actuation, model saturation levels, and help simulate analog actuators with slow frequency responses by using the minimum impulse bit available with the SPHERES hardware.

The inertial and global metrology hardware provide the data accuracy, precision, and observatbility called for in the SPHERES requirements. But this data does not necessarily match the expected metrology information of specific missions. The system requires flexibility to allow scientists to use the data as appropriate for their research. This flexibility comes from the software implementation. First, the metrology system allows scientists to

directly specify the data capture rates of the inertial and global systems independently; the software allows frequency ranges from under 1Hz to up to 1kHz (for the inertial system). Without any special code the SCS allows scientists to model the frequency responses of their sensors. Second, a layer can be created between the standard SPHERES estimator and the scientists use of the states. These modules can simulate sensors not directly available in the SPHERES hardware, such as a star tracker, by modeling the sensor and providing a second state which is used by the scientist's algorithms. In this way scientists can present their algorithms only with the expected available state information, and use the full state calculated by the default estimator as a truth measure to their sensor models.

The limitations of these models lie within the specifications of the SPHERES hardware; the software does not limit the models under the capabilities of the hardware. The propulsion hardware is limited to a frequency of 50Hz; therefore all models will have that maximum frequency. The minimum thruster on-time of 10ms limits the minimum impulse time. Similarly, the maximum sampling rate for the inertial sensors is 1kHz; the maximum rate for the global metrology system is 5Hz. The SCS interfaces with the propulsion system at 1kHz, easily allowing 50Hz operations. The interface with the metrology sensors, both inertial and global, also operates at 1kHz, ensuring that the maximum sampling of the inertial sensors can take place and creating no barriers to access the global system.

> *To better create a physical end-to-end simulation of their system, scientists can create software models of their sensors and actuators which are only limited by the hardware capabilities of the SPHERES satellites but not by the software.*

4.3.4.6 Hardware expansion capabilities

The SPHERES expansion port, presented in Section 4.3.3.2, and the "docking face" directly enable hardware reconfiguration. The primary objective of the expansion port is to support multiple scientists by allowing the addition of specific scientific hardware. The primary objective of the docking face is to enable the demonstration of docking algo-

rithms. But both of these fulfill a second objective: they allow easy manipulation of the hardware to demonstrate increasing complexity of the geometry and/or components.

The expansion port and docking face allow the addition of both passive and active elements with easy. Passive elements can be attached to the docking port by using Velcro in the correct configuration on the additional hardware. This allows the dynamics of the system to change immediately by the addition of different masses. The expansion port allows active elements, be it sensors or actuators, to modify the dynamic behavior of the satellites.

The ability to modify the hardware with active elements depends on the ability of the software to identify those new active components and make use of them. The SCS provides the necessary interfaces so that scientists can access all of the signals available in the expansion port with ease. The SCS always remains as a necessary layer between the hardware and the software. Access to the global bus requires initialization by the SCS; the expansion port global bus data must be accessed through special SCS routines. The SCS also initializes and provides the interfaces for the serial data line of the expansion port. The analog inputs are read automatically by the SCS and made available to scientists via the metrology routines.

> *Passive and active elements can be added via two different locations to implement hardware reconfiguration which increasingly adds complexity to the geometry and dynamics of the satellites.*

4.3.4.7 FLASH memory and bootloader

Previous MIT SSL experiments implemented software reconfiguration [Miller, 1996]; that reconfiguration included the ability to change the state-space matrices of controllers and in some cases the controllers themselves. SPHERES was challenged with allowing high levels of software reconfiguration. The wide range of fields that comprise DSS required that the software reconfiguration not be limited to a specific section of the software, but

rather to a number of major sections. Therefore the SPHERES design implemented a custom bootloader which allows to fully reconfigure the software. As introduced in Section 4.3.2.11, the bootloader allows the operations software to be de-coupled from the hardware implementation. The current implementation of the SCS is not permanently fixed; the SCS can evolve over time, and even different frameworks could be created in parallel to the SCS.

The decision to allow to fully reconfigure the software trades between operational overhead time and flexibility. The decision presents some drawbacks during the development stages of the algorithms. The need to program the software in its entirety adds overhead time to the development process, since even small errors in the code will require to load the full program every time. Yet, the operational plan of SPHERES indicates that during initial development, when operations occur via the simulation or at the MIT SSL, the time to reload a program is not significant. On the other hand, the ability to fully reprogram the satellites is the only way to ensure that the many areas of DSS can be studied over the long term. This ability will enable SPHERES to be used in areas of DSS not currently accounted for by allowing the creation of new threads and interfaces.

To enable full software reconfiguration, the avionics required non-volatile memory which can be overwritten electronically. The selected DSP hardware contains 512kB of FLASH memory onboard. Of that space 256kB are reserved for the board configuration and 34kB for the SPHERES bootloader. Therefore, each satellite provides up to 54k words (216kB) of FLASH memory space for programs; the SCS takes approximately 22k words (88kB), leaving 32k words (128kB) to the scientists. The FLASH memory map is presented in Figure 4.30.

Booting a DSP is a multiple step process. All DSP’s have their own boot program for internal configuration; this boot program is created by Texas Instruments and resides permanently in the DSP chip itself. This process completes in micro seconds. A second boot program configures the SMT375 peripherals so that it can communicate via its TIM 40

01400000 – 0140FFFF	16 kB	Sundance boot loader
01410000 – 0150FFFF	256 kB	FPGA configuration data
01510000 – 0151FFFF	16 kB	FLASH Loader
01520000 – 015FFFFF	224 kB	Application Space
	- 88 kB	- SPHERES Core Services
	- 128 kB	- Scientist code and optional data storage

Figure 4.30 FLASH memory map

standard communication ports, enables the global bus interface, and initializes the interfaces to the internal features of the SMT375. This boot program was custom made for the SPHERES program to minimize the complexity to interact between the SMT375 and the SPHERES peripherals. The SMT375 boot process completes within a few milli seconds.

The SPHERES bootloader is the third boot process. A explained in Section 4.3.2.11, the bootloader is the only mission-critical software in the SPHERES program. Its operation is essential to the success of the mission. This program is loaded by the SMT375 after it is configured. The bootloader first configures the metrology FPGA so that it can communicate with the control panel and all other digital I/O lines. Second, the bootloader configures the three communications micro-controllers to operate at the default data rate of 115.2kbps. Third, the wireless communication channels are set to their default configuration, to ensure that they are functional with the bootloader regardless of any configuration changes by the SCS or other programs (this step takes approximately 2 seconds). These steps leave the satellite in a valid configuration ready for operations.

Next, the bootloader checks the three used communications ports (wireless 868.5MHz and 916.5MHz, and the expansion port serial line) for data commands to initialize the bootloading process as well as the state of the enable button in the SPHERES control panel to determine user override. If data is available or the user forces entry into bootloader mode, it begins to load a new program. The bootloader uses a custom communications protocol with large data packets to minimize overhead; all packet have a two byte checksum. The packets are confirmed at fixed intervals; if a packet is not confirmed the packets are sent again. After loading all the packets, the bootloader calculates a 32bit program checksum

to confirm program integrity. Once a valid program has been loaded a special register in the FLASH memory is enabled, and the boot loader proceeds to load the program.

When no data is available in the communications port the bootloader checks a special register; if the register indicates that no valid program is present it automatically enters into bootloader mode and indicates a "no program" condition in the control panel.

When a valid program is present and the bootloader has no other pending actions, it loads the program into memory. Loading the program takes a few milli seconds. If the program is a standard SCS application, the program first configures the SPHERES peripherals for use with the SCS standard interfaces, and then runs the SAT INIT process and enters the idle mode described in Section 4.3.1.4. Table 4.11 summarizes the full SPHERES boot process.

A custom bootloader allows the full software of a SPHERES satellite to be reprogrammed and stored in FLASH. The bootloader automatically starts an existing program if no command is received to load a new program.

TABLE 4.11 SPHERES bootloading process

Step	Process	Time	Enables
1	C6701 Boot Process	μs	C6701 core, memory interfaces, and embedded peripherals
2	SMT375 FPGA/ DSP configuration	ms	SMT375 communications ports, global bus interface, LED's, DSP/FLASH interface
3	SPHERES Bootloader	2s	SPHERES FPGA (metrology, propulsion, internal beacon, housekeeping, and control panel I/ O's), DR200x wireless communications
4	SCS Sat Init	ms	API to SPHERES peripherals, TDMA wireless communications, metrology configuration, background telemetry, DSP/BIOS real-time environment, satellite logical identity

4.4 Summary

Table 4.12 summarizes the characteristics of SPHERES which enable it to fulfill the MIT SSL Laboratory Design Philosophy. A thorough operations plan and carefully designed software and avionics (enabling families of tests, easy repetitions, separation from safety controls, and quick data feedback) *facilitates the iterative research process*. The visual nature of SPHERES further helps to speed up iterations.

The design of the nano-satellite hardware *supports experiments*, satisfying all but one features called upon by the philosophy. The metrology and communications systems enhance *data collection*. The 32-bit DSP ensures *data precision* throughout all data processing. The test management plan and location specific GUI's facilitate *repetitions*. The re-supply of consumables provides system *reliability,* enables *extended investigations,* and creates a *risk-tolerant environment.* The use of more units than essentially necessary and the fact that software cannot cause a critical failure also create a *risk-tolerant environment*.

The Guest Scientist Program, through its logistics, the SPHERES Core Software, simulation, and standard science libraries, together with the flexible schedule of SPHERES, directly *supports multiple investigators.* The Expansion port further enhances the ability to *support multiple investigators* by allows investigator-specific hardware to be used in experiments. The portability of SPHERES increases the number of operational locations for the facility, such that *multiple investigators* can use the hardware in the preferred locations for their specific science.

The implementation of the SPHERES nano-satellites as a standard satellite bus provides a perfect example of the development of *generic equipment*, while at the same time creating a *physical end-to-end simulation* of a spacecraft. At the same time the implementation of docking-specific equipment, the SPHERES hardware also demonstrates the use of *specific equipment*. The physical nature of the SPHERES satellites, with their ability to fully simulate complex DSS missions, creates a realistic *physical end-to-end simulation* of expected missions. The generic operating system and software interface to the SPHERES sensors

TABLE 4.12 Summary

SPHERES Characteristic	Sub-System					Feature Group			
	Avionics	Communications	Software	Operations	System	Iterative Research	Support of Experiments	Multiple Investigators	Reconfig. & Modularity
Multi-layered operations plan				✓	✓	✓		✓	
Continuous visual feedback					✓	✓			
Families of tests			✓			✓			
Easy repetition of tests			✓			✓	✓		
Direct link to ISS data transfer system		✓				✓	✓		
De-coupling of SW from NASA safety controls	✓		✓			✓		✓	
Layered Metrology System	✓		✓				✓	✓	
Flexible communications		✓	✓				✓	✓	
Full data storage		✓					✓		
32-bit floating point DSP	✓						✓		
Redundant communications channels		✓					✓		
Test management and synchronization			✓	✓			✓		
Location specific GUI's				✓			✓	✓	
Re-supply of consumables	✓				✓	✓	✓		
Operations with three satellites					✓		✓		
Software cannot cause a critical failure	✓		✓		✓		✓		
Guest Scientist Program				✓				✓	
Expansion Port	✓							✓	
Portability					✓			✓	
Schedule flexibility				✓				✓	
Implementation of a satellite bus	✓	✓	✓		✓				✓
Science specific equipment	✓				✓			✓	✓
Generic operating system			✓						✓
Physical simulation of space environment	✓	✓			✓				✓
Software interface to sensors and actuators	✓		✓			✓		✓	✓
Hardware expansion capabilities	✓								✓
FLASH memory and bootloader	✓				✓	✓			✓

and actuators provide a *modular* software platform which provides *generic* command and data handling functions while allowing *software reconfiguration* to meet the *specific* needs of scientists. Lastly, SPHERES enables both *hardware* and *software reconfiguration* through its expansion port, use of FLASH memory, and the development of a bootloader which works independently of the SPHERES Core Software applications.

By meeting practically all the features of the MIT SSL Laboratory Design Philosophy and operating making the correct use of the resources of multiple facilities (SSL Lab, KC-135, and ISS), SPHERES is more than a testbed for formation flight, it is a laboratory for DSS. Recall the definition of a laboratory (page 60): *a place providing opportunity for experimentation, observation, or practice in a field of study.* SPHERES does provide the opportunity for experimentation, as it facilitates the iterative research process. Further, SPHERES supports the research of multiple scientists whom can work on different areas of DSS, enabling the practice in a field of study.

Lessons were learned from following the MIT SSL Laboratory Design Philosophy in the development of the SPHERES laboratory. These lessons are presented in the following chapter as the Design Principles for the Development of Space Technology Maturation Laboratories.