# A University-developed Comprehensive Open-architecture Space Mission Operations System (COSMOS) to Operate Multiple Space Vehicles

Trevor C. Sorensen[1], Eric J. Pilger[2], Mark S. Wood[3], Miguel A. Nunes[4]
*Hawaii Space Flight Laboratory, University of Hawaii, Honolulu, HI, 96822*

Bruce D. Yost[5]
*NASA Ames Research Center, Moffett Field, CA 94035*

**The Hawaii Space Flight Laboratory (HSFL) at the University of Hawaii at Manoa has a 3-year NASA EPSCoR project to develop a comprehensive system that is designed primarily to support the operations of one or more small spacecraft, but can also perform an important role in the design, development, and testing phases of spacecraft missions. This set of mission operations tools operate within an architecture named COSMOS (Comprehensive Open-architecture Space Mission Operations Support). COSMOS is particularly suited for small operations teams with a very limited development and operations budget, such as universities. The COSMOS project is being performed with the collaboration of the NASA Ames Research Center. COSMOS is a suite of software tools and hardware that enables the Mission Operations Team to interface with the spacecraft, ground control network, payload and other customers in order to perform the complete set of mission operations functions.**
**The basic philosophy of the COSMOS architecture is that its elements (tools and other programs) will be easy to port to a new location and to modify for operating with new spacecraft. This is possible by using an "open architecture" that includes the source code of its major elements, but it is also designed to accept external modules as plug-ins through standard, well-defined interfaces. Although COSMOS was initially designed to operate multiple small spacecraft, during its development other applications for its use have been identified and in some cases prototyped. These include lunar missions with monitoring and control of both the lander and rover, and remote monitor and control of space-related facilities. MOST is currently being modified in collaboration with .the Space Dynamics Laboratory, to support the DICE mission, and with the NASA ARC, to support Phone Sat and other missions.**

## I. Introduction

The Hawaii Space Flight Laboratory (HSFL) was established at the University of Hawaii at Manoa in 2007 for two primary purposes: (1) to educate students and help prepare them to enter the technical workforce, and (2) to help establish a viable space industry that will benefit the State of Hawaii. HSFL is currently developing a solid-propellant launch vehicle capable of placing small satellite (< 300 kg) into low Earth orbit (LEO) and various satellites from CubeSat (1-kg) to micro-satellite size (80kg).[1] HSFL is installing the infrastructure and facilities to support space missions, such as clean rooms for integration and testing, ground stations, mission operations center, and simulators/test beds. It is the goal of HSFL to provide full life cycle mission operations support for our space missions. This requires the use of specialized software to develop and sustain mission operations. Based on a trade study and analysis of the various software packages and systems available, we determined that none met the full functionality and flexibility we desired while staying with our budget constraints. The primary author was experienced with mission operations for many space missions and had developed some tools where were successfully used in a low-Earth orbit (LEO) mission and a lunar mission[2] and would be very suited for HSFL's

[1] Professor and Project Manager, AIAA Fellow
[2] Flight Software Engineer
[3] Instrumentation Engineer
[4] Graduate Student, AIAA Student Member
[5] Program Manager

needs with some modifications and improvements. From this was born the idea of developing a comprehensive system of software and hardware to efficiently support mission operations for multiple spacecraft, especially small satellites. We decided that such a system should have a framework for plugging in external applications and tools that would enable them to work with the whole system (plug'and'play attribute). There was also an obvious advantage to have the system be open architecture, including software, which would enable a large community of users to help rapidly develop new features and enhancements to the system that would be difficult in a closed system. This new system and architecture was named the Comprehensive Open-architecture Space Mission Operations System (COSMOS) and was the basis for a successful proposal to NASA in 2010, resulting in a 3-year NASA EPSCoR Grant to develop and deploy COSMOS. The grant started in 2010 and ends in 2013. NASA Ames Research Center is the primary collaborative NASA center. We are at the mid-point of this project and although the design of COSMOS has been presented in previous papers,[3,4] this paper looks at the evolution and development of COSMOS and how it has been adapted to support different types of missions, including some types that were not foreseen when the COSMOS project was proposed and initiated.

## II. COSMOS Overview

COSMOS is a framework of software and hardware elements that addresses all phases of a spacecraft life cycle; Design, Development, Implementation and Operations. It provides elements for the creation of simulators, test beds, and flight and operations software, all fully interactive. Reference 3 describes the functional architecture and operational concept of COSMOS. The guiding principle of the COSMOS suite is that it will be easy to port to different locations, and to configure for different spacecraft. The following tools are used to perform the major functions of mission operations for COSMOS:

1) Mission Planning and Scheduling Tool (MPST)
2) Mission Operations and Support Tool (MOST)
3) Test Bed Control Tool (TBCT)
4) Ground Segment Control Tool (GSCT)
5) Data Management Tool (DMT)
6) COSMOS Executive Operator (CEO)
7) Various Analysis and Support Tools

To achieve this end, COSMOS is based on a limited number of key design elements, put together in a layered approach (Fig. 1). These key elements are based as much as possible on existing protocols and approaches. The foundation of COSMOS consists of a set of libraries supporting the various functionalities available in the suite. This includes mathematical functions, and orbital and coordinate calculations, protocol support, and hardware and simulation support for the Operations Test Bed (which will help test and verify command loads before they are uploaded to the satellites).

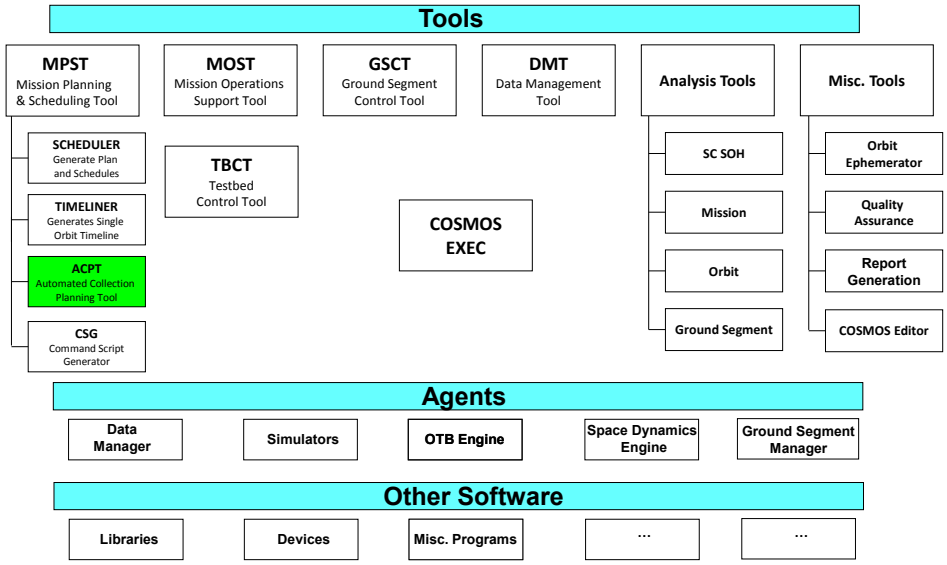


**Figure 1. COSMOS Software Elements**

The COSMOS project is an effort to create a complete open environment for the design, development, and operation of small spacecraft. The broad philosophical approach can be found in that paper, although the concepts and implementation are continuing to mature and evolve as COSMOS becomes operational. What follows is a synopsis of the major fundamental software elements, or building blocks, of COSMOS, with specific details that will be important to discussions in the later sections.

## B. Protocols

### *1. Unified Name Space*

As originally implemented, the COSMOS Name Space was a completely flat mapping of names to single values. In this approach, each name represented a single string or number. The meaning, units and data type of each value were predefined. An attempt was made to further simplify things by assuming that values, such as position, which could be defined in multiple systems, were always represented in one predefined system. Since no arrays were used, column (and row) positions were indicated with digits embedded in the name. A name like "panel_01_point_01" represented point \#1 of panel \#1.

Each member of the Name Space was then mapped to a unique location in memory; the memory locations being drawn from a globally accessible storage space. This Name Space Map was stored as a simple list of entries, allowing either forwards or backwards translation between COSMOS Name and memory storage.

This approach has proven quite flexible. However, it has also caused a number of problems, especially as the space has grown in size. This led to a number of changes, which are detailed in Section III.

### *2. COSMOS Subset of JavaScript Object Notation (COSMOS JSON)*

In keeping with the original definition of the Name Space, the original COSMOS JSON included only strings and numbers. Arrays and complex objects were not supported. The purpose behind this was to support the completely flat Name Space, and to simplify the mapping of names to variables in memory. It has subsequently become necessary to expand this usage, both in support of the Name Space changes noted above, and to more accurately represent values in memory.

### *3. Agent Communications Protocol*

The original implementation of Agents in COSMOS called for them to send out a *Heartbeat* packet of information at regular intervals to a predefined COSMOS Multicast address. This packet would contain the IP address of the machine on which the Agent was running; a unique *Request* port, on which the Agent would listen for incoming requests; the size of the buffer available for Requests, and any resulting Responses; the name of the Agent; and the period of the heartbeat, in seconds. While this basic approach has proven successful and remains mostly unchanged, certain supporting elements are still in flux. The best mechanism for determining the IP address and Port of the Agent, the relative merits of Broadcast versus Multicast, and the use of IP itself are still being discussed.

### *4. Network Protocols*

The original design for COSMOS specified the use of IPv4 UDP protocols in both a Unicast and Multicast mode. The Lightweight Communications and Marshalling (LCM) protocol was then used on top of this to provide communications and data translation between Agents. It soon became clear that LCM incurred a large amount of programming overhead, and was somewhat redundant in light of changes we made to our use of JSON and the Name Space. We are still also trying to define the best protocol for use between the satellite and the ground. While the NORM protocol is still in the running, we are also giving serious consideration to the Cubesat Space Protocol (CSP). CSP is interesting both because it might serve well as a protocol for all segments, and because it already has a well defined file transfer protocol, a needed feature of COSMOS that has yet to be defined.

### *5. Configuration Files*

A number of configuration files are used to communicate with MOST, or COSMOS in general, allowing them to be fine-tuned to a specific mission. Foremost of these is the Satellite Description File, named "satellite.ini", which is a JSON description of all the static elements of the vehicle. It starts with a description of the physical Parts, followed by a listing of any Components that are then tied to specific Parts, followed by specific Devices that are tied to Components. Parts include any physical qualities. Components describe general electrical qualities. Device for each Device Type describe qualities unique to that device. The goal of the Satellite Description File is to provide sufficient detail to reasonably simulate all aspects of the vehicle. A separate Ground Station File, named "groundstation.ini", is also created for each mission so that the Ground Segment for that mission can be simulated. This is also a JSON string.

### C. Support Hardware

It has been the ongoing goal of the COSMOS project to develop hardware simulators for all the components to be found in a spacecraft. Extensive progress has been made on the groundwork for this effort with basic off-the-shelf hardware and processors.

### D. Support Software

The suite of libraries, as initially laid out, has not changed significantly, other than to have numerous bugs identified and eliminated. Two additional libraries proved necessary; one in support of enhanced mechanisms for satellite description, called *satlib*, and a second for support of OpenGL graphics, called *graphlib*.

### E. Agents

Only a cursory description of Agents was provided in Ref. 3, so a more detailed explanation will be given here. As described above, Agents make their presence known through a *Heartbeat*. Any client wishing to communicate with an Agent listens to the COSMOS Multicast address until it either receives a *Heartbeat* from the Agent it is waiting for, or it times out. Once it acquires the desired Agent, it sends a *Request* to the IP Address and Port taken from the *Heartbeat*. It then waits for a *Response* on that same IP address and Port until it receives one, or times out. All *Requests* and *Responses* are in plain ASCII.

### F. Tools

Extensive work has been done on MOST since it was first described Ref. 4. The latest version set up for a 3-U CubeSat is shown in Fig. 2. Quite a few features that were merely described have now been implemented, and significant changes to both internal workings, and features have been driven by our work on the various Test Cases that are listed in the next section. The need for an overarching Tool to control all the Entities functioning in a COSMOS system has also become clear. We have dubbed this the COSMOS Executive Operator (CEO).

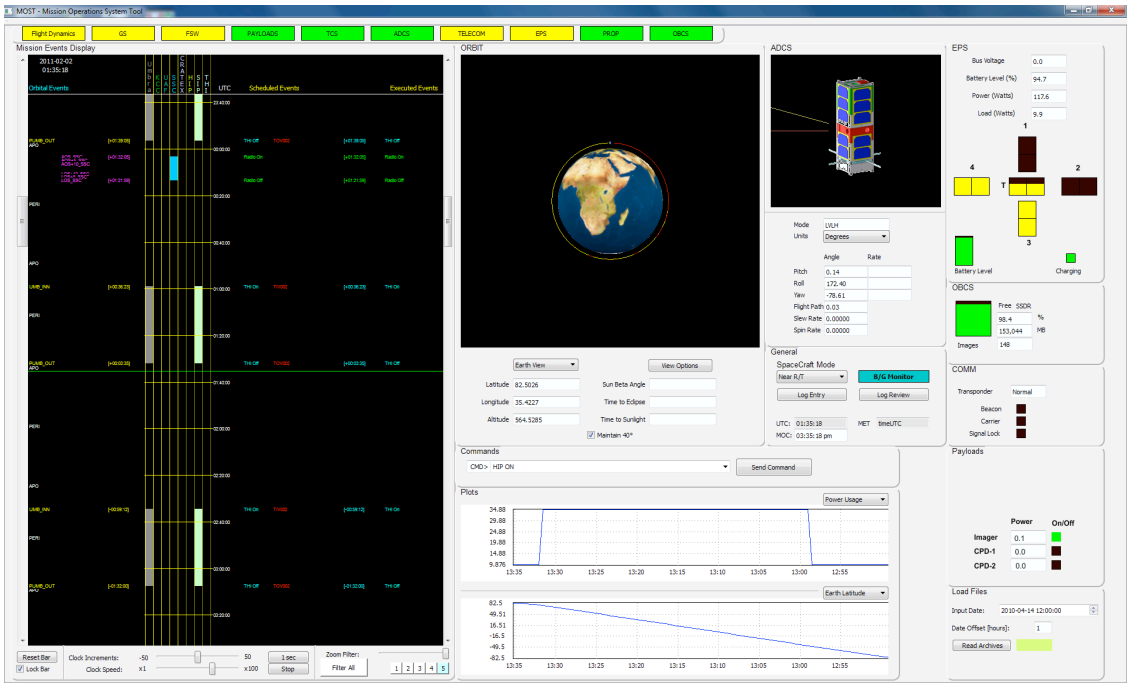


**Figure 2. MOST Overview Display for 3-U CubeSat**

MOST is written in C++ using the Qt (pronounced "cute") compiler.[5] A modular approach using Qt "widgets" has been used with idea of making MOST easily reconfigurable using configuration files. Qt compiles on Windows, Linux, and Mac, so MOST will be able to run on any of these systems. The Mission Operation System Tool (MOST) displays information about a satellite. It is the main interface between a satellite and an operator. It shows position and attitude related to an object it is orbiting. It shows telemetry data and past and present orbital and

spacecraft events. Each subsection of MOST is contained in a "Widget". Each widget contains information about a specific aspect of the spacecraft. The widgets presently used in MOST are widgets that contain the time, Warning Buttons, Mission Events Display (MED), orbit, ADCS, EPS, OBCS, Comm, Payloads, TimeLine Plots, and a command line to send commands to the spacecraft.

## III. Modifying MOST for New Spacecraft

### A. Test Cases

Since 2010 we have had a number of opportunities to fit MOST, and the underlying COSMOS framework, to a variety of real-world examples. This has allowed us to:

- Identify problems with our existing code base and make appropriate corrections.
- Spur the creation of additions and enhancements to both the existing tools, like MOST, and to the whole philosophy of COSMOS.
- Identify areas of future concern that will need solutions, and spur the invention of future features.

This subsection provides detail on these test cases, while the following subsections describe the changes that were brought about in both our code and our overall approach to COSMOS development as a result.

*1. Summer High School Internships*

We were host to 6 high school juniors through a UH College of Engineering intern program. With this group, we were able to start putting in to use some of the basic components of the OTB and COSMOS software. For the OTB, we laid down the foundational use of Arduino processors, coupled with XBee radios. For COSMOS software, we developed the first test version of an Interpreter Agent to interface a non-COSMOS data stream.

*2. NASA ARC Phone Sat*

In collaboration with the NASA ARC, we are developing a version of MOST that can be used with a personal receive only ground station to monitor passes of Phone Sat from any location..

*3. NASA ARC EtherSat*

We have arranged with the NASA ARC to use MOST, and the COSMOS system, to support the EtherSat mission of 70+ Phone Sats. This is driving our design and development of the COSMOS CEO.

*4. NASA ARC Mission Control Tools (MCT)*

We are studying the similarities and differences between MCT and COSMOS, with the intention of creating a support framework between these two environments. Interesting possibilities exist both for giving access to the COSMOS structure to MCT, and for embedding elements of MCT in MOST and other tools.

*5. USU DICE Mission*

We have agreed to modify MOST to provide shadow operations for this mission, converting their mission telemetry to a form that MOST can understand and then allowing them to view it within MOST. We are currently creating a satellite description for this mission and have started addressing conversion of the test data set they have already given us.

*6. CABLE*

Our most detailed test case, and the one that has driven the greatest number of advances to date, has been the Canadian American British Lunar Expedition (CABLE). This collaborative effort, detailed in Ref. 6, allowed us to extend MOST to demonstrate its utility in support of a multivehicle, planetary mission that also included a rover. This mission, with three vehicles, thrusters, ground based operations, on multiple planetary bodies drove numerous changes. We created a number of enhancements to MOST, as well as defining features we would like to have in later versions of COSMOS. Examples of the work done on MOST in support of CABLE are shown in Section III.C.

*7. UH Undergraduate Outreach*

As part of the educational aspect of HSFL in general and COSMOS specifically, we have created a number of projects for UH students who are NASA Space Grant Fellows, and for a Mechanical Engineering class. The Space

Grant Fellows have allowed us to push forward a variety of projects related to OTB hardware and flight related concepts. The Mechanical Engineering class provided much needed field experience.

*8. COSMOS Itself*

The next big Tool level project in the COSMOS suite is the COSMOS Executive Operator (CEO). Always part of the planning for later stages, development of this software has risen in priority by the funding of the EtherSat mission by NASA. Consideration of what this software will need has driven a number of changes to the existing software and plans for future enhancements.

**B. Problem Areas Identified**

Perhaps the most important outcome of our various test cases has been the identification of problems that were going to hold us back in either the short or long term. These problems are detailed below along with any immediate response. Any long-range changes to our philosophy are then covered in subsequent subsections.

*1. Code Errors*

One of the first things to be identified, from the very first test case, were the various errors that existed in the code we had already written. In some cases the errors were subtle issues, while in others it was impossible to move ahead until flaws in either logic or implementation had been fixed. This subsection details these various problem areas.

Basic Functionality

An immediate benefit of the work one of our summer interns did with the Interpreter Agent was the discovery of numerous bugs and logical flaws in our support libraries. In order to deal with this in a systematic way, we set our first student hire to work developing exhaustive test suites for the various libraries. He has completed work on the Math Library, and has begun work on the Conversion Library. Our long range plans are to have a similar test program for each library in the COSMOS suite.

Thread Safe Code

As we continued support for Hawaiisat-1, and moved in to developing support for missions like PhoneSat and DICE, we started creating fully functioning agents. It became clear at this point that more care had to be taken with the thread safe nature of the support libraries. This was especially true for the JSON library, where the same engine was used by multiple threads for communication with the outside world. It quickly became apparent that the functions to create JSON strings were stepping on each other. This has been resolved for the moment by asking for a user-supplied buffer. This is not a perfect solution, since it still depends on the user being responsible for avoiding problems, but it did allow us to move ahead.

In the long term we will analyze this issue in more detail, and also consider other areas in the code that could be open to conflict. The end goal will be to relieve end users from having to spend too much time worrying about system programming types of issues.

Transfer Buffer Size

The Agent and JSON support libraries are all about the creation and transfer of buffers. As we moved into extensive use of these libraries through development work for the CABLE mission, we started running into problems with both relative buffer sizes and absolute buffer limits. The JSON support routines were coded with dynamically expanding buffers; and both sets of routines keep careful track of the sizes of buffers. However, it is still an issue when one agent that can only handle a 4KB buffer is handed a 40KB buffer. This problem only became worse as we started making use of agents dispersed over the network and started running in to problems with routers and bridges.

In the short term we juggled increased buffer sizes and decreased data transmission to stay within the workable limits of our immediate environment. In the long term we are considering changes that will allow the fragmentation of buffers that will decrease the magnitude of this problem and increase the scope of networks that we can travel over.

*2. Hard-Coded Behaviors*

As with any software project, the requirement to get some initial functionality in place, combined with deadlines, led to a version of MOST that was hand-tailored to our original mission, HawaiiSat-1. As we began to adapt this to other missions, especially the multi-vehicle CABLE mission, we found ourselves rewriting the same program multiple times.

Integrating the code of several different software developers, while keeping all the functionality, can be an issue. Some sort of version control was essential, so we adopted code development using Subversion. Simply adopting a version control solution was not enough, however, and we have subsequently adopted conventions for usage that optimizes the effectiveness of this approach. This approach saw its greatest challenge during our work for CABLE, and we now feel confident working multiple versions of the software with multiple programmers simultaneously.

Hand Crafted User Interfaces

We have modified MOST several times in order to support different missions. For each mission we have identified parts of the code that have previously been hard-coded, and have modified the code to allow the user interface to be created "on the fly" using a configuration file. The goal is to allow MOST to be configurable for many different missions with little or no modification of the code. For example the Mission Events Display (MED) had some mission specific subsections hard-coded. During our last update, we have removed any mission specific coding from the MED. The MED now is configured by the mission's configuration file.

Hard Coded Name Space

The COSMOS Name Space is proving to be quite useful, and will prove to be a critical element in the openness and flexibility of COSMOS. At the same time, it has had to go through a number of iterations since it was first imagined with more changes in the offing. Chief among these was a major modification of the naming rules. As originally defined, all numerical values were to have their own names. However, this soon proved to be quite cumbersome, as well as causing problems with associated but disconnected values. We therefore made a major rule change to the naming convention, and now assign names to larger structures, such as *position*.

This major change of naming rules, as well as some other minor changes, has also emphasized the problems with hard coding Name Space support. As a result, we are formulating some future mechanism by which we could define the Name Space in some configuration file from which the support code would be automatically generated.

*3. Performance Issues*

Part of our last update included changing the way MOST handled orbital events and spacecraft events for the "Mission Events Display" (MED). Previously, we had two files, and correspondingly two software routines that handled them. One for each type of event. We combined these two into one events file, and now have one routine to handle this data. This speeds up the process of displaying the data, but reduces the amount of code needed.

Name Space Mapping

Another aspect of the Name Space mapping that we soon discovered needed work was related to its performance. Setting up, and then using the map, was taking far too much time. As a result, we modified the code from a straight linear search to a hashed database approach. This has dramatically decreased set up time, as well as increased access speed.

In order for a Universal Name Space to work properly, there must always be a physical structure in memory to which to map each name. The strict requirement of having a physical location for each name can lead to a number of unfortunate consequences. Chief among these is the large amount of space that must be reserved for every possible instance of a name. This can be mitigated somewhat by limiting the number of instances of a name, but then leads to the second problem of a limited Name Space. We have mitigated this problem somewhat in the current version of MOST by being more efficient in how we define things. In the long run we will have to do something more dramatic. Our ideas for this will be covered in the subsection on future enhancements.

Graphical Widget Reuse

As we introduced more graphical elements into MOST, it became clear that we needed to be far more efficient in both our coding, and use of plots. We found ourselves being inefficient both during coding, through rewriting essentially the same code in different parts of MOST, and during execution, through unnecessary recalculations of drawing commands. We have taken a step back and are rethinking our graphics environment to identify common elements and views. We have already implemented draw lists for commonly used elements such as satellite models and planetary bodies, as well as using standard widgets whose behavior is modified through parameters, thereby making the same code reusable. In future we hope to create a simpler, more efficient, yet flexible approach to all our graphical views.

### C. Current Enhancements and Features

*1. Improved Name Space and COSMOS JSON*

We have relaxed some of the limitations on both the Name Space, and the JSON that represents it. The use of a flat Name Space proved to be too limiting, so we have expanded it to allow complex structures built upon the simple elements. In order to support the various types already defined in COSMOS, we have had to provide support for

arrays and complex objects in JSON. We now support most of the vector and matrix type defined in mathlib, as well as a variety of position and attitude types defined in convertlib. At the same time we have embraced a broader range of names, allowing names that reflect the exact system in which a set of values, such as position, were defined. The mapping of JSON to memory now includes automatic conversions to other systems where it makes sense.

*2. Satellite Description Generator*

The more vehicles we embraced, the clearer it became that we could not continue generating the Satellite Description File, satellite.ini by hand. In support of the CABLE mission, we developed a number of tools to help us with this process. The list of Parts, Components and Devices is nothing more than a simple database, and we developed a set of linked spreadsheets that help us manage the information. This can then be dumped to a series of files that are read by a COSMOS program that spits them back out as a valid JSON strings and writes them to satellite.ini and groundstation.ini.

A greater challenge has been the generation of the Parts to put in the database. We have developed a process using a simple model developed in SolidWorks that is then exported as VMRL, and finally converted with a second COSMOS program to a Parts description.

*3. Additional Views*

In order to support the variety of vehicles and different planetary bodies represented by the CABLE mission we were required to create some new variations on our existing Vehicle and Orbital views. New Vehicle views includes: a Chase view which follows close behind the vehicle, either in orbit, or in the case of the Rover, on the ground; and a Lander view which remains at a fixed point on the ground, and follows the vehicle. The Orbital view was modified to support different planetary bodies as its center of focus. Finally, we developed a two-dimensional Map view showing the horizontal placement of the vehicle. Examples of these various views of MOST running the CABLE mission are shown in Figures 3-9.

*4. Static Versus Dynamic Data*

As we started using full sets of telemetry data, and to consider what maximum frequency of data we would like to handle in the future, we began to search for ways to be more memory efficient. We quickly realized that the arrangement of values in memory that made sense for a simple simulation was quite wasteful of memory when transferred to an archival array of thousands of values. As a result we split the storage in to static and dynamic values and now store them in different parts of the global data structure.

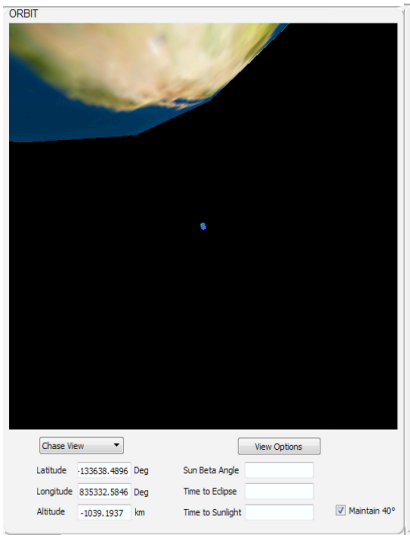


**Figure 3. Chase View**

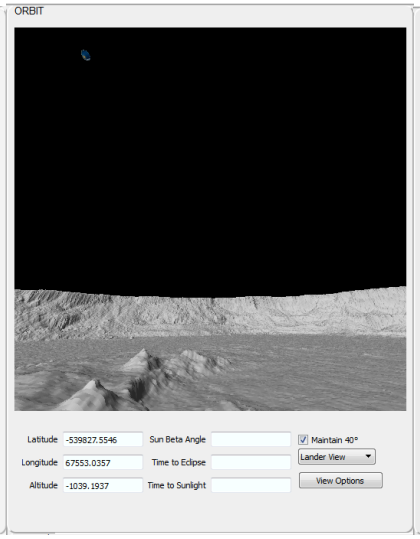


**Figure 4. Lander View**

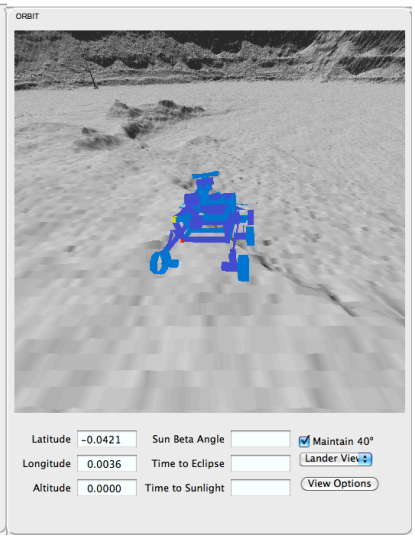


**Figure 5. Lander View - Rover**

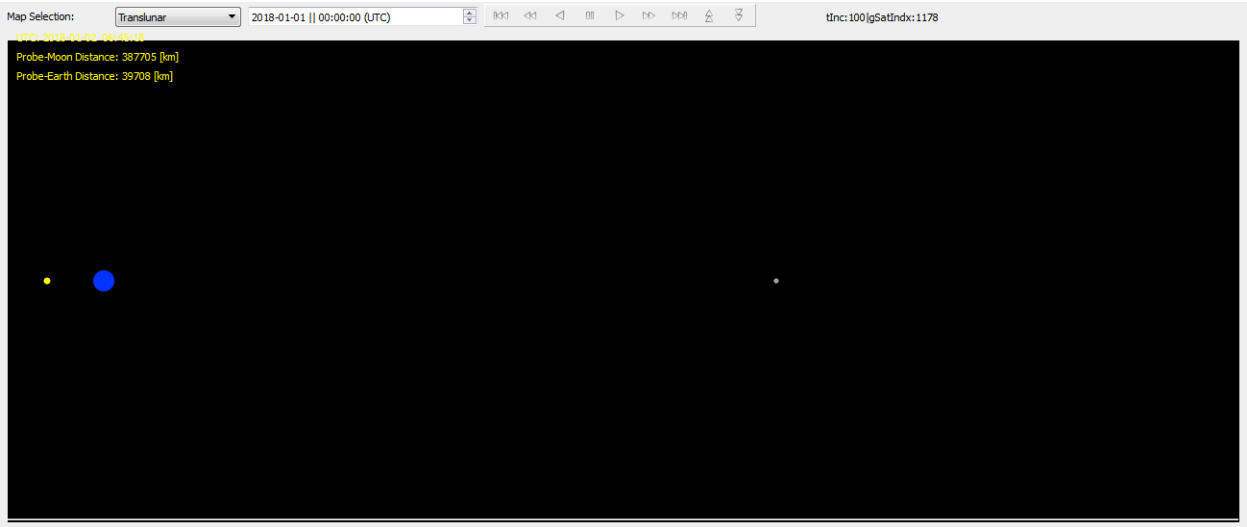


**Figure 8. Cis Lunar Map**

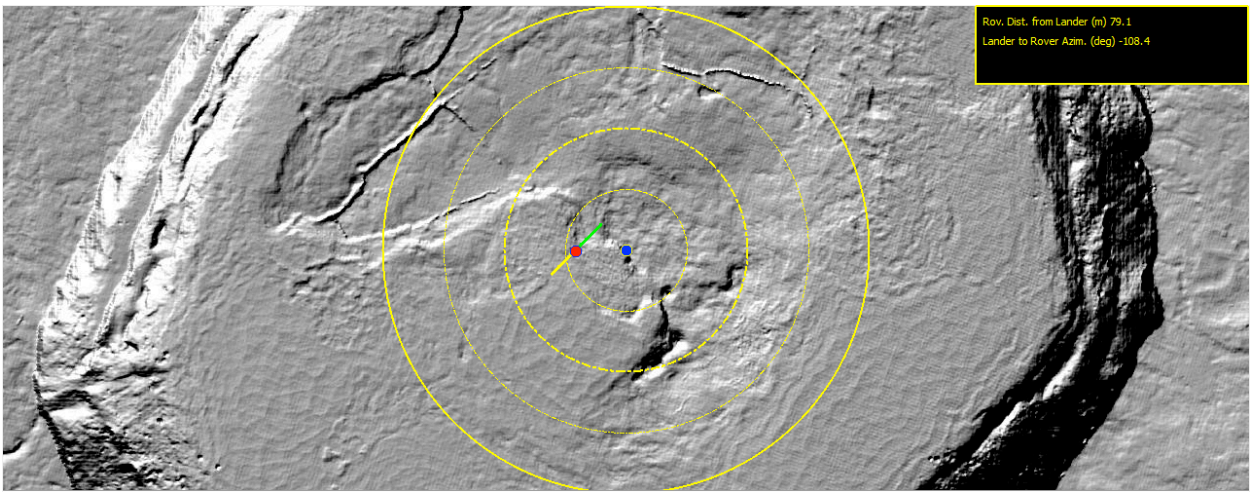


**Figure 9. Surface Map for Rover Operations**

*5. MOST Features Enabled*

It has always been a goal of MOST development to support not only archival data, but also instantaneous real time data. We also planned to allow simple spacecraft commanding, also in real time. In support of the class project of a Mechanical Engineering class, we were able to develop both these features. MOST can now receive SOH updates dynamically, as transmitted by a COSMOS SOH Agent. The commanding feature of MOST now interfaces via the network to an Agent that can accept Requests. Several predefined Requests are shown on a drop-down menu, or a Request may be typed in manually. It is then sent to the appropriate Agent via the Agent Communications Protocol.

The MOST caution and warning lights have also been enabled to indicate the warning level. Normal operation is Green, cautionary condition is Yellow, and an anomalous condition is Red. The indicator lights now also show a pattern indicator to the right of each button that changes in accordance with the warning level. This allows a color-blind person to determine the warning level.

*6. MOST Engine*

Through development of the CEO it has been clear that we needed to separate the data management functions of MOST from the graphical display. We are therefore in the process of separating MOST into a user interface component, and a second Agent (the "MOST Engine") that runs as a background task. The engine constantly monitors telemetry data, and does any required calculations. The user interface portion of MOST communicates

with the Engine via the Agent Communications Protocol, and receives and displays the data. The "MOST Engine" can constantly monitor the mission, even if no user interface is active. It will generate alerts and have the capability to e-mail, or text important developments or problems to key personnel.

Separating the computationally intensive aspects of MOST into its own Agent will allow the CEO much better access to, and control of multiple missions. As the number of missions grows, the number of MOST Engines can easily be scaled up through Grid computing solutions, such as Condor. It also allows more flexible access to specific mission data from multiple graphical interfaces simultaneously.

### D. Future Enhancements and Features

For every improvement added to COSMOS, our work with the above Test Cases has generated new ideas for future improvements. The following sections are a short list of the ones we plan to work on first.

1. *Memory Friendly COSMOS*

We current assign the global data structure statically, which has the dual disadvantage of reserving large amounts of memory, even when it is not needed, and at the same time limiting the number of various items to preset maximum values. We are working on a mechanism that will allow us to assign the space dynamically, while retaining some of the current convenience of being able to set one fixed structure equal to another.

*2. Fragmentable JSON*

Upper limits to the size of packets, and their fragmenting as they pass through different networks, has become a problem as COSMOS has grown. We are developing a modification to our use of JSON that will allow JSON strings to be as long as required and yet survice over any network link.

The proposed approach makes what, in standard JSON, will actually be a series of single element JSON strings. This will then allow the string to be broken between any two elements if, at any time during its passage, it comes across a bottleneck smaller than its current size. When split, each new element will be given its own Timestamp so that all will arrive at their destination with proper time information.

*3. Standardized Widgets and User Interface Files*

Most Qt applications get their graphical functionality through the combination of compiled code, and a "ui" file on disk. It has always been the goal of COSMOS to allow reconfiguration without recompilation through the dynamic loading of these files. By changing the file, you can change the interface without having to recompile the code. A limitation to this, however, is that the code must reference the "ui" file. Names in code and file must match, or the connection is lost. Furthermore, as new subsystems are introduced, they will have neither code nor "ui" file, and it would be nice for them to have some default behavior driven solely by the satellite.ini file.

The solution we have proposed is to develop matched generic "ui" writing and loading routines A portion of the global data structure will be chosen to represent, either by preagreement, or by communication, say through the Name Space. The writing routine will then create a generic "ui" file for these elements, and the loading program will load this "ui" and populate it using the agreed upon labels. We have begun work on a set of support routines for generating the XML necessary to create “ui” files. Once that is completed, we will move on the some form of basic automatic widget.

*4. Self-Contained MOST*

The PhoneSat team has asked for a version of MOST that could be used in combination with a small portable ground station for receive-only monitoring. We are combining a copy of MOST, and a MOST Engine, all in a virtual linux machine that could run on any other OS.

*5. System Nodes and Entities*

As we stretch the limits of COSMOS, it becomes clear that it has a potential for far more things than just spacecraft, or even vehicles. In an effort to develop a broader concept in which satellites are just a subset, we have working on the idea of a COSMOS Node or Entity. In this new context, Nodes would be assemblages of related Agents, such as for a Ground Station, or a Mission Operations Center. Entities would be things like users that existed in the system, but were not necessarily represented by an Agent. In this scenario, communications would be between Entity:Agent combinations, as for instance KCCGroundStation:Executive and HMOC:DataManager.

*6. Multiple Nodes per Mission*

COSMOS employs a powerful feature to support more than one Vehicle, (or Node) per mission. We are building in the ability to recognize multiple Nodes within a single mission, allowing rovers to be aware of their landers, or satellites in a swarm to be aware of each other.

*7. Changing MOST Clock*

The MOST "master clock" determines what time is displayed as the "present" time. It has been using the built-in QT QDateTime structure to keep track of time. This has had some confusing issues, and has limited the resolution of the master clock to one second. This will be changed from the QDateTime structure to a fractional Julian Day values. This will allow higher resolution of time, and fix some of the issues that have caused confusion.

## IV. Operations Test Bed and Simulators

The COSMOS Operations Test Bed (OTB) is coherent with the open-source system architecture that integrates hardware and software components to operate a low cost Satellite System Simulator (e.g. FlatSat) which can be integrated into the Mission Operations Center setup for command scripting testing, personnel training, mission rehearsals and anomaly resolution. The OTB has tools for satellite technology integration and development that allows for cheaper satellite subsystem integration and testing. The OTB tools are based on COTS that are affordable to university laboratories while some tools are being developed under the COSMOS project using proven standards and made available to the small satellite community. The OTB is part of the four major processes in mission operations that are supported by COSMOS, namely the Mission Planning and Scheduling, Real-time contact operations, mission analysis, and anomaly resolution. This testbed is being designed to accommodate multiple spacecraft testing by reutilization of the same tool that are highly configurable, and so generalizable for the different satellites. Another important design feature of the OTB is that is capable of scaling for testing satellite constellations.

One important aspect of the OTB is that it makes possible to provide an interface with different satellite hardware and simulators that are needed to make the global testing procedure for different missions. This platform also allows the mission segment functional simulation and mission rehearsals from the command sequence to the software and hardware performance. One more important aspect to note is that the OTB is being designed so that it may be remotely operated, allowing people from different remote locations use this same setup to help in their satellite development or mission operations

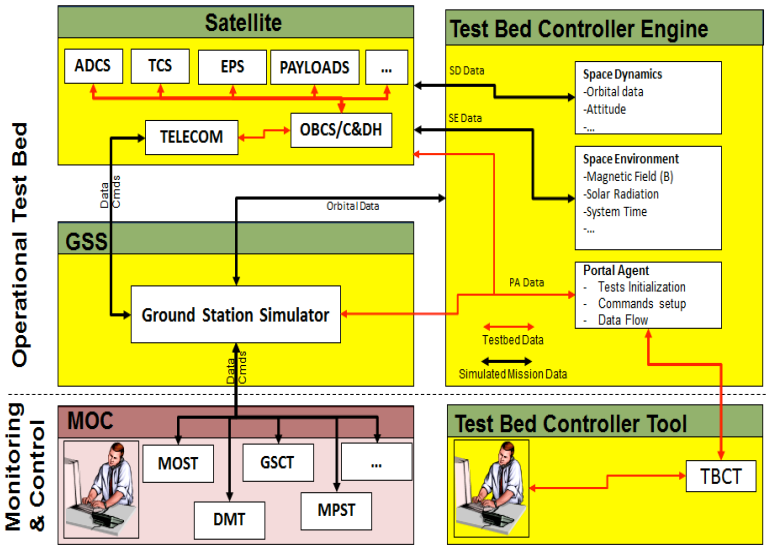


**Figure 10. OTB Architecture**

To completely operate the OTB its setup must integrate six main constituents: (1) The actual Mission Operations Center (MOC) control tool, or MOST; (2) the Ground Station Simulator (GSS); (3) the Satellite System and Subsystem Simulator (SSS); (4) the Test Bed engine (TBE); (5) The test bed controller tool (TBCT); and (6) the Test bed controller user interface. This segmentation is expressed in Figure 10.

The MOC System Simulator allows the end user to conduct the near real-time spacecraft system and subsystems testing and operational activities, including mission planning; assessment and maintenance; instrument health monitoring; and communications, command and control function. The integral part of the MOC System Simulator is MOST, which is one of the two interface tools between the OTB and the end user. Reference 3 expands on some of these functionalities of the OTB.

Open source frameworks targeted to be used in real-time systems are considered as primary resources for the development of the OTB (such as ACE, LCM, etc.). The Satellite System and Subsystem Simulator platform integrates all the satellite subsystems to be operated (e.g. ADCS, TCS, EPS, Telecom, etc.). These can be either fully operational with the engineering model hardware components or else software simulated if the hardware components are not readily available. Based on the Test Bed Engine, it supports full propagation of the test satellite's conditions, in both real and faster than real time. Figure 11 shows a subsystem of the OTB being tested for development of the On Board Computer System for the HawaiiSat-1 microsatellite. Figure 12 shows the HiakaSat full-scale mockup on top of the air bearing testbed setup. This testbed was developed with the help of Daniel Wukelic, a Space Grant fellowship student working with HSFL. This will be used to test mostly the Attitude Determination and Control Subsystem by using MOST to connect through a GSS to the mockup. This part of the OTB is comprehensive in its nature because it is the culmination of many previous OTB subsystem tests after which the system is integrated and then tested before launch.

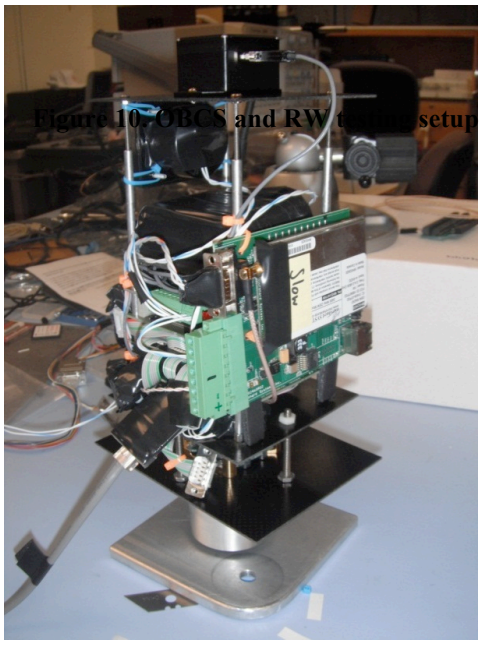

**Figure 11. Single-Axis ADCS Test Bed**

The Test Bed Dynamics Engine provides a software simulated space environment to the OTB to allow a more realistic operation of the whole platform. The dynamics engine also controls the different hardware and software configurations in the satellite system simulator and allows the tuning and mixing of signals and interrupts, adding noise and possible failure modes. All this is done either controlled by the controller user interface or a scripting sequence. The Test Bed Control Tool (TBCT) is an application to support the experimental set up for the OTB architecture. The TBCT interfaces with the GSS, the satellite system, the Test Bed Engine and the end user. It allows initializing and controlling the satellite system platform and the Test Bed Engine according to the user decisions or scripting.

The user interface control tool is software like MOST to operate and change the OTB parameters and testing sequences.

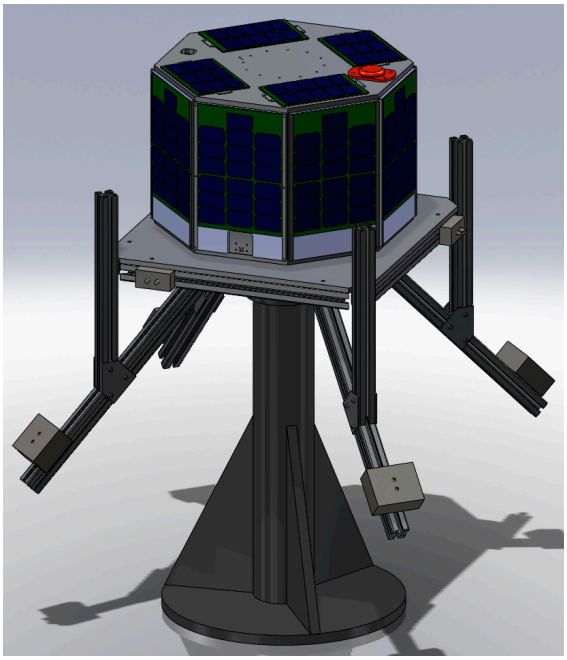

**Figure 12. HiakaSat Mockup on air bearing testbed**

The COSMOS OTB can incorporate different hardware parts that are made available for testing and experimentation. These components can include common sensors, actuators and other hardware systems that are common for satellite integration. Other specific features of the OTB include: calibration and testing of hardware components; integration of software tools for hardware simulation; subsystem validation & monitoring; subsystems interaction & dynamics monitoring; pseudo-environment input; anomaly resolution; measurable performance: like pointing, timing, speed, fast, power, etc.; remote control of the OTB using scripts; near real time testing and simulations; mission training and rehearsal; trending and analysis; system operation rehearsals and simulations with statistical analysis (e.g. Monte Carlo, Dead Reckoning); operability with different standard software development tools and languages: MATLAB, LabView, Phyton, C/C++, and/or other engineering COTS software utility tools; support the development and operational test for different satellites.

## VI. COSMOS Executive Operator

COSMOS is being designed to handle multiple satellites on multiple missions. When the number involved are only a few (maybe less than a couple dozen), COSMOS can handle multiple missions in a single MOC, with each satellite having its own session of the major COSMOS tools, either on the same or different consoles. If the facility resources permit, it would be simpler and easier to dedicate one console per satellite, with another to host the GSCT, one for the DMT, and a top-level coordinating console running the COSMOS Executive program.

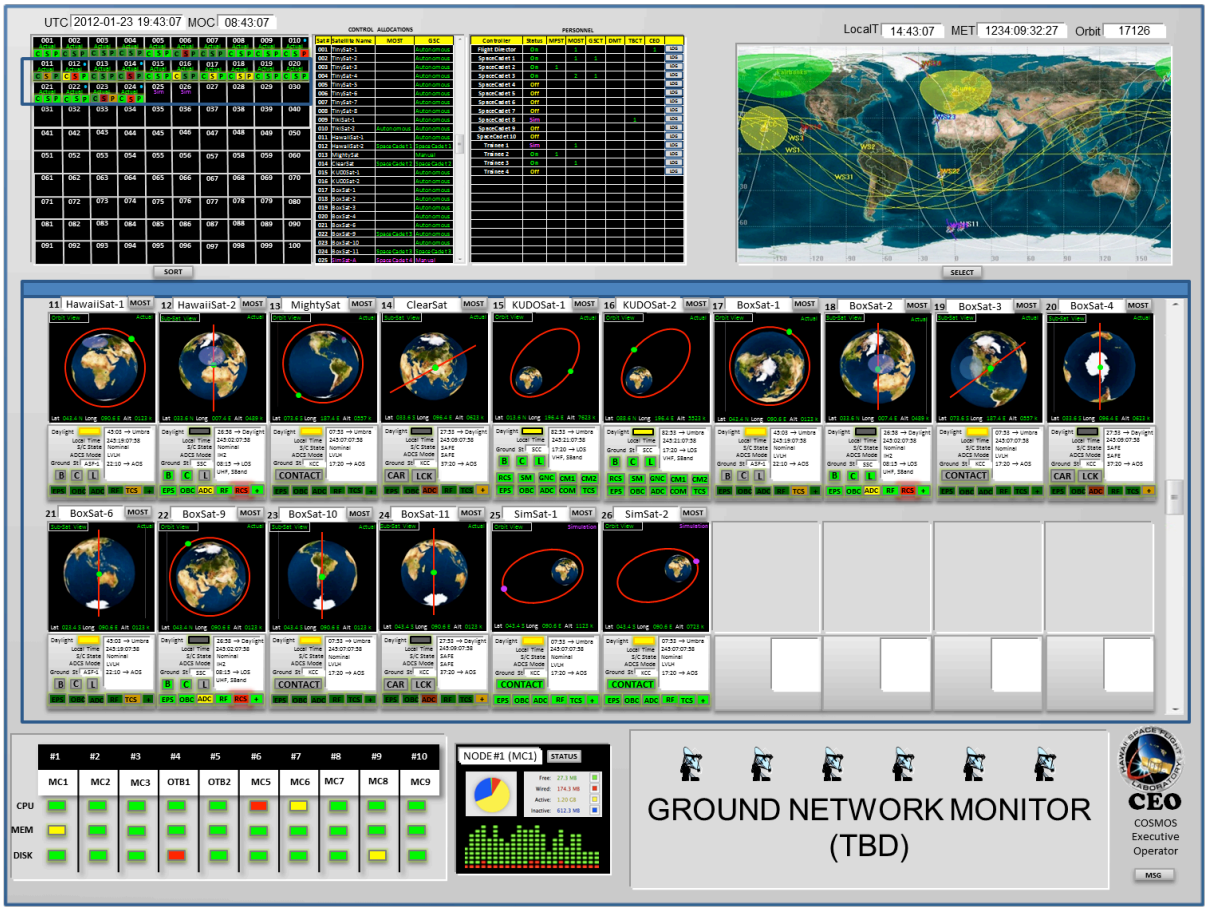


**Figure 13. Mockup CEO Display** – *the display is showing 20 satellites, but Ground Network Status is not yet operational*

The COSMOS Executive Operator (CEO) program Provides situational awareness of multiple spacecraft or simulated spacecraft simultaneously. In its initial implementation CEO can handle 100 spacecraft, but there is no reason it cannot be modified to handle more. There are three different selectable levels of monitoring:

1) Low – Spacecraft ID, status of spacecraft, status of Payload, Ground Station contact status
2) Medium – shows orbit position and data, day/umbra status, Ground Station contact status, status of subsystems, spacecraft or attitude/pointing modes, etc.
3) High – Similar to main display of MOST giving detailed information

The CEO collects the information from the MOST Engines running with the Data Management System (one MOST Engine is running for each satellite being monitored). These MOST Engines send their data out through the DMS to any application that wants it, such as a session of MOST or the CEO.

CEO also provides situational awareness of the Ground Segment, including the status and operations of all the ground stations in the network. It can launch the GSCT for more detailed information or commanding of the Ground Segment.

Besides the external elements of COSMOS outside the MOC, the CEO also provides management of the MOC operations. It monitors allocation of COSMOS tools to spacecraft, monitors personnel utilization, access console logs (current or archived), and communicate with one or more MOC positions directly. The CEO monitors the COSMOS system performance, such as the console computers performance and utilization, and the status of the

COSMOS tools and the data flow between the COSMOS elements. The CEO can also launch any of the COSMOS tools, such as the MPST, MOST, GSCT, DMT, TBCT, and analysis tools.

The design of the CEO is completed and coding is just starting at the time of this conference. It is expected to have a prototype version running by the end of 2012, and basic operational version of CEO finished in mid-2013.

## VII. Conclusion

The COSMOS project has almost reached the half-way point in the three-grant provided by NASA. Some of the major elements have been developed and demonstrated in a prototype form, and it will soon be deployed to support various current or upcoming space missions, initially in the monitor-only mode, but eventually with full command capability as well.

During the first half of the project, the software building blocks needed to develop the high-level operations tools have been developed and mostly completed, although improvements are being made as our experience with the system grows. The lynchpin of COSMOS and the tool most developed, MOST, has shown its versatility in being adapted from supporting just Earth-orbiting missions, to prototyping a version to support lunar missions, even extending its capabilities to supporting control of rovers. This was beyond the original range of applications that were considered at the outset of the project, but once the flexibility and capabilities of MOST were demonstrated, we came to realize that its application potential was much larger than realized. The Canadian firm, MPB Communications, has a contract with the Canadian Space Agency to build and demonstrate a miniature planetary rover using Earth-analogous terrain. They have requested a version of MOST to support this effort. The OTB is the other major element of COSMOS that is well under development and is currently being used in the development and testing of HSFL satellites.

During the remainder of the project period our primary goals are to deploy operational versions of MOST at various MOCs, including at NASA Ames Research Center and the USU Space Dynamics Laboratory. We also will be working on the development of the remaining tools, especially the MPST and the GSCT. The CEO is vital to controlling multiple spacecraft or objects simultaneously, and development of it has started but is in the early stages. A fully functional basic version of COSMOS should be operational by the end of the NASA grant, at the end of August, 2013. However, COSMOS will continue to be developed within HSFL and with the aid of our government, university, and industrial partners, who will constitute the COSMOS community.

## Acknowledgments

We would like to thank the hard work put into this project by the following members of the COSMOS team: Faculty and Staff: Harold Gabreil, Daniel Watanabe; Undergraduates: Erik Wessel, April Vogt; High School Summer Interns: Mathew Esporas, Max Dylan Matsuda-Hirata, Daron Lee, Erin Main, Jennifer Nishida , Grant Takara

## References

[1]Sorensen, T.C., French, L., Chan, J.K., Doi, William K., Gregory, E.D., Kobyashi, M.H., Lee-Ho, Z.K., Nunes, M.A., Pilger, E.J., Yamura, R.A., Yoneshige, L.K., "Hawai□iSat-1: Development Of A University Microsatellite For Testing a Thermal Hyperspectral Imager," AIAA-2010-8922, AIAA SPACE Conference, Anaheim, CA, 30 Aug – 2 Sept, 2010.

[2]Sorensen, T.C., Tran, T.T., Geldzahler, B.J., Horan D.M., Prescott, R.J., "Effective Science Mission Planning and Operations - The Clementine Approach," Paper RAL.GS.31, 1st Annual Reducing the Cost of Space Ground Systems and Operations Symposium, Rutherford-Appleton Laboratories, June 1995.

[3]Sorensen, T.C., Pilger, E.J., Wood, M.S., Nunes, M.A., "Development of a Comprehensive Mission Operations System Designed to Operate Multiple Small Satellites," SSC11-IX-3, 25th Annual AIAA/USU Conference on Small Satellites, Logan, UT, 8-11 August, 2011.

[4]Sorensen, T.C., Pilger, E.J., Wood, M.S., Gregory, E.D., Nunes, M.A., "Development of the Mission Operations Support Tool (MOST)," AIAA 2010-2230, SpaceOps 2010 Conference, Huntsville, AL, 25-30 April, 2010.

[5]Qt Technical Overview Whitepaper - http://qt.nokia.com/files/pdf/qt-4.4-whitepaper pg. 41).

[6]Kruzelecky. R.V., Haddad, E., Nakhaei, A., Jamroz, W., Cloutis, E., Sorensen, T., Mouginis-Mark, P., Shan, J., Hamel, J., de Lafontaine, J., Barnet, M., Teti, F., Ghafoor, N., "CABLE Canadian American British Lunar Explorer," GLEX-2012.03.1.5x12366, Global Space Exploration Conference, Washington D.C., May, 2012..