A Testbed for Evaluating Lunar Habitat Autonomy Architectures

David Kortenkamp[1], Michel Izygon[2], Dennis Lawler[3], Debra Schreckenghost[1], R. Peter Bonasso[1], Lui Wang[3], and Kriss Kennedy[4]

[1]*TRACLabs Inc., Houston TX 77058*
[2]*Tietronix Inc., Houston TX 77058*
[3]*NASA Johnson Space Center/ER2 Houston TX 77058*
[4]*NASA Johnson Space Center/EA3 Houston TX 77058*
281-461-7884 korten@traclabs.com

Abstract. A lunar outpost will involve a habitat with an integrated set of hardware and software that will maintain a safe environment for human activities. There is a desire for a paradigm shift whereby crew will be the primary mission operators, not ground controllers. There will also be significant periods when the outpost is uncrewed. This will require that significant automation software be resident in the habitat to maintain all system functions and respond to faults. JSC is developing a testbed to allow for early testing and evaluation of different autonomy architectures. This will allow evaluation of different software configurations in order to: 1) understand different operational concepts; 2) assess the impact of failures and perturbations on the system; and 3) mitigate software and hardware integration risks. The testbed will provide an environment in which habitat hardware simulations can interact with autonomous control software. Faults can be injected into the simulations and different mission scenarios can be scripted. The testbed allows for logging, replaying and re-initializing mission scenarios. An initial testbed configuration has been developed by combining an existing life support simulation and an existing simulation of the space station power distribution system. Results from this initial configuration will be presented along with suggested requirements and designs for the incremental development of a more sophisticated lunar habitat testbed.

Keywords: Lunar Outpost, Simulations, Automation
PACS: 07.05.Dz

INTRODUCTION

NASA is planning to build a permanent lunar outpost over the next two decades. The lunar outpost will likely be built from many different modules that will be flown separately and connected on the lunar surface. The lunar outpost will have many interacting subsystems that will need to be monitored and controlled. These subsystems include Environmental Control and Life Support Systems (ECLSS) and Electrical Power Systems (EPS), which are the current focus of the testbed. Each of these is composed of their own subsystems. For example, ECLSS contains water reclamation, air revitalization, thermal control and solid waste disposal. These subsystems themselves have their own subsystems. And habitat modules may have their own ECLSS and EPS that then need to be merged with those of new modules that arrive later.

Habitats on lunar surfaces will have complex interactions between the crew, the environment, chemical and biological life support systems and possibly crops such as tomatoes and lettuce. These habitats will be significantly more complex than the International Space Station (ISS) because resupply will be limited, resulting in more complicated, regenerative life support systems. Also, lunar habitats will need to support extravehicular activities (EVA) on a routine, probably daily basis.

NASA also has the goal of lunar outposts having significant crew autonomy from ground controllers and the goal of reducing the ground control staffing requirements. There will be times when the lunar outpost is uncrewed. Thus, automation will be necessary to control the various systems composing the lunar outpost. This automation will be at

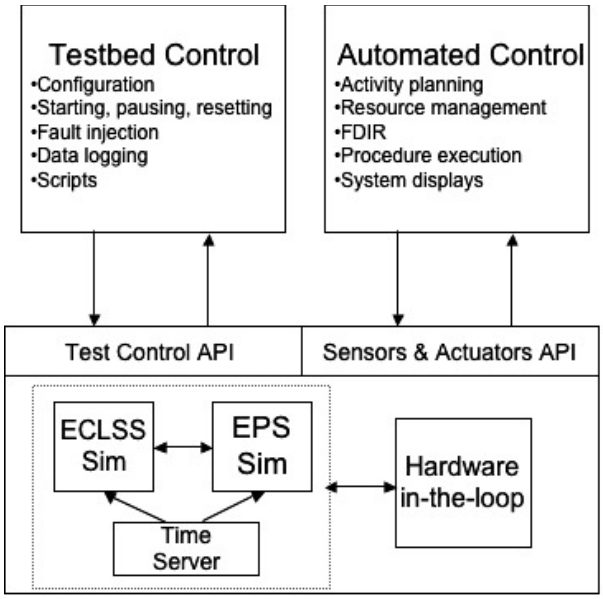

FIGURE 1: The Components of the Habitat Testbed.

a level of complexity that has not yet been demonstrated in space systems. The number of interacting systems, the incremental build-up and the time-delay between the Moon and the Earth all impose significant challenges to automation software. In order to explore these issues and evaluate different automation technologies, approaches and architectures a testbed is necessary. Initially the testbed will contain software simulations that provide a "stand-in" for hardware. As hardware is developed it can be included in the testbed. The objective of this paper is to introduce the motivations, requirements and initial capabilities of a lunar habitat ground testbed so that the community can coordinate and collaborate with NASA in testing different lunar outpost configurations and operational concepts.

Similar habitat testbeds have been built previously – some with hardware and some only in software. Hardware testbeds include several at NASA including the Lunar-Mars Life Support Test Program (Miller, Edeen and Sirko, 1993) and the BioPlex (Tri 1999), which was never completed, but a full software simulation was developed by NASA (Finn, 1999). A large close-loop life support testbed called Closed Ecology Experiment Facilities (CEEF) has been developed and maintained by the Japanese for many years (Miyajima et al, 2006). Software simulations of life support systems have been developed over the last several years. Some are static, such as the ALSSAT tool (Yeh et al, 2004) and others are dynamic such as (Rodriguez, Kang and Ting, 2000). Of course, BioSphere was a well-publicized testbed for closed-loop, regenerative life support and the Mars Society maintains several analog sites including the Mars Desert Research Station (MDRS) in Utah. The latter does not focus on life support but instead on extravehicular activities (EVA) and geological science. Several habitat testbeds are currently in the planning stages, including a small chamber being developed by Paragon Space Development Corporation (Linrud et al, 2007) and another in Spain by the European Space Agency (ESA) (Mas, 2007).

Figure 1 shows the testbed components. The testbed consists of some software simulations (in the current case an ECLSS and an EPS simulation but there could be others) that exchange data and are synchronized by a time server. The testbed also optionally consists of hardware-in-the-loop that replaces some software simulations with actual hardware that exchanges data with the software simulations. For example, an ECLSS simulation could be integrated with actual water reclamation hardware to provide a complete habitat testbed before all the hardware is ready. The simulations and hardware have two application programmers interfaces (API) that communicate with external processes. The first API is for testbed control and the second API is for system control via sensors and actuators. A testbed control program is responsible for configuring and controlling any testbed activities either via scripts or via a graphical user interface (GUI). It uses the test control API. An automated control system is responsible for controlling the systems in the testbed. It uses the sensors and actuators API. When the system is deployed the testbed control program no longer exists, but the automated control system provides control of the deployed

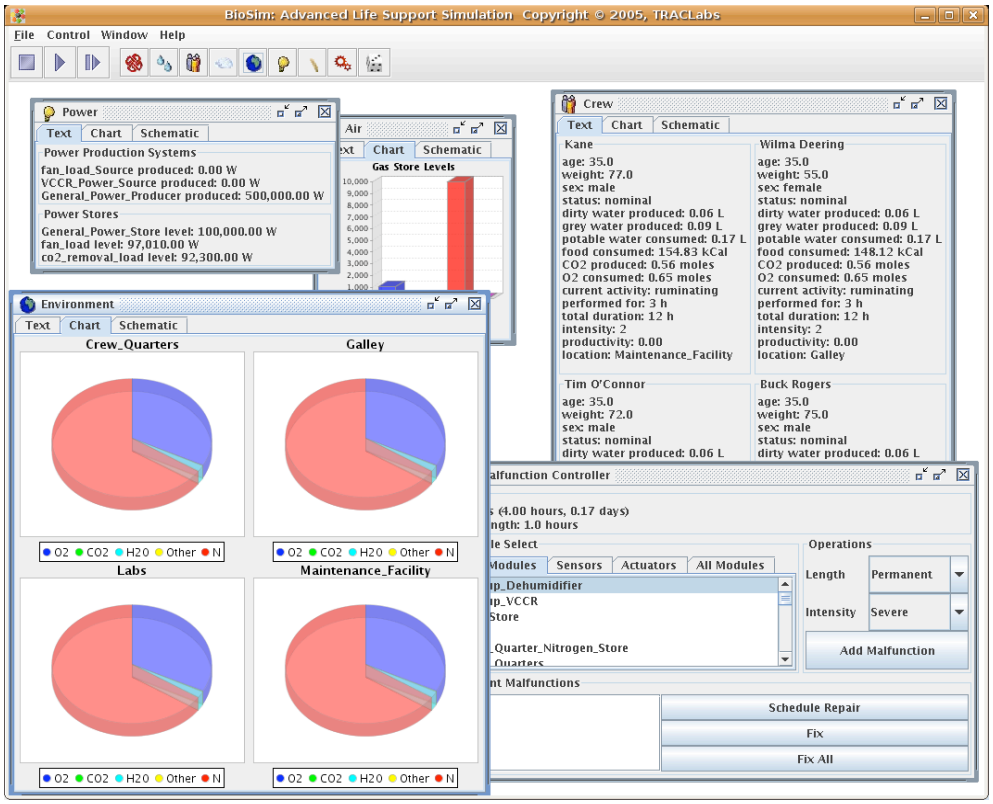

FIGURE 2: A View of the ECLSS Simulation User Interface.

hardware via the same sensor and actuator API. This allows seamless transition from testbed experimentation to deployed systems. The rest of the paper will discuss each of the testbed components.

SOFTWARE SIMULATIONS

Two systems of a lunar outpost are being simulated: ECLSS and EPS. The two simulations are executing separately and connected via the internet to exchange data. In particular, the ECLSS simulation is receiving its power from the EPS and the EPS is receiving power requirements from the ECLSS. First, the ECLSS simulation is discussed, then the EPS simulation, and finally the integration of the two.

ECLSS Simulation

The lunar outpost ECLSS simulation is derived from the BioSim models developed over the past several years (Kortenkamp and Bell, 2003). BioSim is a discrete-event simulation of a space habitat that models each of the life support components of a habitat as processes that consume certain resources and produce other resources. For example, the water recovery system model consumes dirty water and power and produces potable water.

Crew and EVA

The most important module is the crew module. The crew module is implemented using models described by (Goudarzi and Ting, 1999). The number, gender, age and weight of the crew are settable as input parameters. The crew cycles through a set of activities (sleep, maintenance, recreation, etc.). As they do so they consume O_2 food and water and produce CO_2, dirty water and solid waste. The amount of resources consumed and produced varies according to crew member attributes and their activities. The crew module is connected to a crew environment that contains an atmosphere that they breathe. The initial size and gas composition (percentages of O_2, CO_2, H_2, O_2 and inert gases) are input parameters. As the simulation progresses the composition of gases in the atmosphere changes.

Extravehicular activities (EVA) involve crew members leaving the habitat for variable periods of time. While outside the habitat the crew members no longer place a load on the habitat life support systems, but instead need alternate life support systems. This simulation is implemented by spawning a "mini" habitat with parameters settable by the user. This mini-habitat exists for the duration of the EVA. The mini-habitat has its own environment, stores, recycling systems, etc. During the mini-habitat's existence the crew member would breath her air, drink her water, etc. and would not be drawing from the original, larger habitat. When the EVA is over, the mini-habitat is merged back with the original habitat.

Air Revitalization

An air revitalization module provides breathable air to the crew environment. The CO_2 Removal System (CRS) takes air from the crew atmosphere, removes CO_2, and returns the remaining air mixture. The removed CO_2 is collected in a CO_2 store. The CO_2 Reduction Assembly (CDRA) takes CO_2 from the store and reacts it with H_2, making H_2O and CH_4. The methane (CH_4) is sent to the Methane Reduction System (MRS) and the H_2O is passed to the third system, the Oxygen Generation System (OGS), which breaks down the H_2O into H_2 and O_2. The O_2 is saved in an O_2 store and the H_2 is returned to the CDRA. The MRS splits the CH_4 into carbon, which is sent to the solid waste store and hydrogen, which is used by the CDRA. The air revitalization module may take H_2 from the potable water store for use with the OGS, or it may put H_2O into the waste water store. In addition, a dehumidified removes excess water (caused by human breathing) from the air and adds it to the waste water store. Injectors are available to take gases from the stores and inject them into the atmosphere.

Water Reclamation

A water recovery module consumes dirty and grey water and produces potable water. Grey water has no biological waste and is produced by washing and similar activities. A Cascade Distiller System (CDS) is modeled as the water reclamation system. The CDS is currently undergoing tests at NASA Johnson Space Center (Lubman, 2007) and there are data from those tests to build models. For a CDS the more power applied to the system the more potable water produced. The water reclamation system also includes a post-processing system that polishes water to make it potable.

Habitat Configuration

BioSim is easily configurable via an eXtensible Markup Language (XML) file. This file specifies all parameters of the habitat including the volume and pressure of the habitat modules, the sizing of the life support components, the number, ages and genders of the crew members, the crew and EVA schedule, etc. For this project BioSim was configured to contain four habitat modules. One contains the ARS. Another is the crew sleeping quarters. A third is a galley and living area and contains the WRS. The fourth is a laboratory module with a backup CO_2 removal system. Modules one and four contain airlocks each 8 m^3. Each module has a volume of 27 m^3 at 85kPa with 34% O_2 concentration. Fans exchange air amongst all of the modules. Crew schedules are also configurable. Assumed was a crew of four with a workday of eight hours of sleep, two crew members with eight hours of normal metabolic activity and eight hours of EVA and two crew members with fourteen hours of normal metabolic activity and two hours of exercise (e.g., high metabolic activity). The weekends have slightly more sleep, no EVAs and a few hours of exercise.

EPS Simulation

For the EPS simulation a simple finite state machine (FSM) based simulation was developed. This type of simulator keeps track of the states of the different elements of a system, receives the commands from a procedure executor and updates the system states accordingly. Each system FSM simulator also triggers subsequent transitions in other systems and can execute some scripts. The State Machine formalism adopted by the Universal Modeling Language (UML) community was used. This is an object-based variant of the Harel state charts concept. The UML state machine formalism provides support for concepts such as simple, nested and composite state, transitions, guard conditions, entry and exit action, activities, join and fork. As the Lunar Habitat EPS system is not yet defined at the level of details needed to develop a full blown simulator, the FSM approach provides the required behavior and allows us to start testing some operational scenarios. This type of simulator enables a continuous modification and upgrade of the models as more details are provided during the design phase.

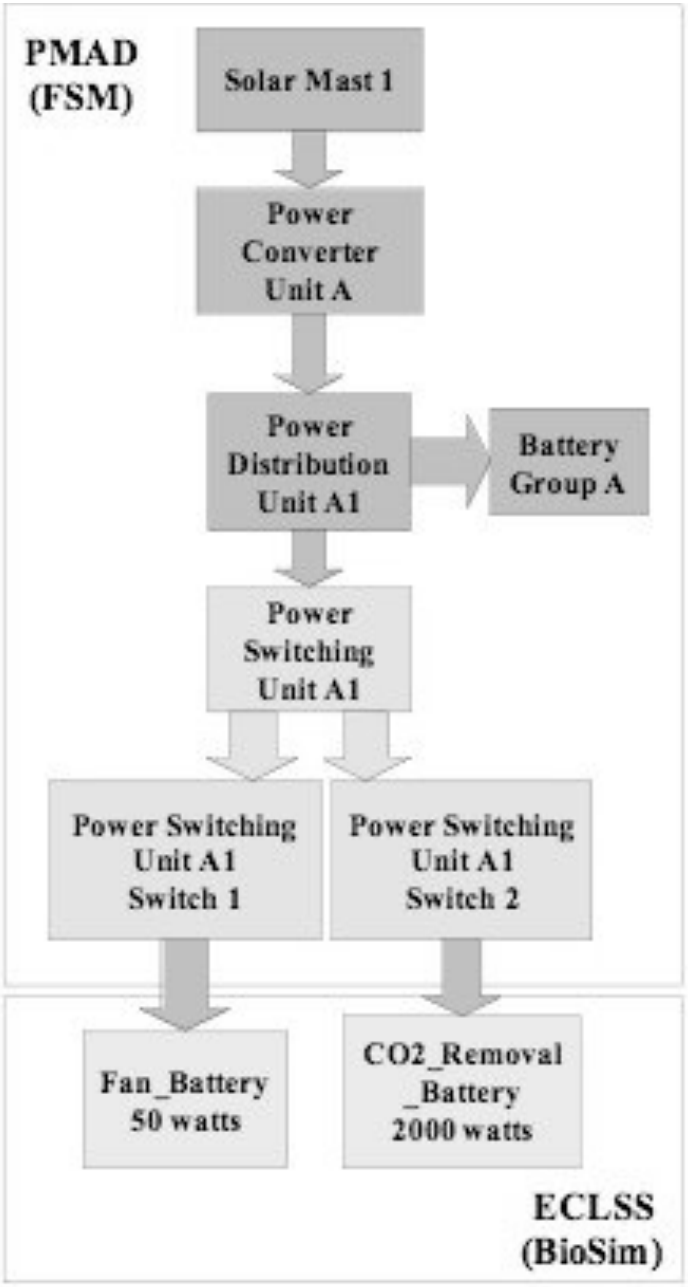

Figure 3: EPS Components in the Software Simulation and Their Connections to the ECLSS Simulation.

In order to develop this simulator, the current early habitat designs from the Lunar Architecture Team (LAT) studies, previous experience with the development of similar systems for the International Space Station EPS, and a given set of scenarios that includes the occurrence of failures, were used. Based on these current Minihab Power Architecture Design (MPAD) concepts, developers captured the components involved, and assigned to each component the states they can be in, the events that trigger the corresponding transitions and the interrelations between these components.

Figure 3 shows a representation of the EPS components modeled using the FSM approach. It contains the Solar Array unit, a Power converter unit, a Power distribution unit which distributes the power either to a Battery for storage or to a Power switching unit for distribution to the ECLSS loads through two switches. The EPS simulation also provides the capability to inject some failures, such as an overvoltage Switch Trip, that trigger state transitions.

Integration

Figure 3 shows the integration of the ECLSS simulation and the EPS simulation. At each time tick the ECLSS simulation asks the EPS simulation for power for a fan and for a CO_2 removal system. The EPS returns the power, which is added to the battery store attached to the device. If a failure is inserted into the EPS it may return zero power to the ECLSS simulation.

TESTBED CONTROL

There is an infrastructure for configuring and managing the testbed. This infrastructure allows for choosing the components of the testbed, setting up their initial states, controlling them, inserting faults into the testbed and logging data from the testbed. This infrastructure is provided by an application programmers interface (API) that describes software functions that can be called on the testbed. The API makes it easy to integrate new software or hardware into the testbed as long as it implements the API. In the testbed, the API is implemented in the Common Object Request Broker Architecture (CORBA), which is an industry-standard architecture for distributed commanding. CORBA supports most major programming languages (e.g., C++, Java, etc.) allowing for diverse applications to connect to the testbed. The infrastructure also provides a set of configuration and scripting files represented in extensible Markup Language (XML). The XML files allow for pre-scripting of testbed activities.

Set-up and Configuration

A testbed may consist of many different components that are interconnected. In the example presented in this paper we have an ECLSS and an EPS component. There may be others (e.g., thermal). There are also different configurations of the ECLSS and EPS components that might be desired (e.g., number of crew members, pressure of modules, nuclear vs. solar, etc.). In addition, there might be hardware-in-the-loop that needs to interface with software simulations during testbed operations. Finally, the testbed components may have initial states that need to be set. The testbed configuration is stored as an XML file that is read by programs that construct the testbed instance. There is also a CORBA API that allows for some changes to the configuration while the simulation runs. Currently there is no graphical user interface to allow testbed users to change the configuration, but that is a logical next step.

Control of the Testbed

There are two kinds of control that are necessary for the testbed. First, software simulations can be started, stopped and paused. If hardware is in-the-loop then this is not always possible. The ECLSS and EPS simulations have start, stop, pause and reset commands that are available to external processes via an API. These commands are also available via an API script. The second kind of control is over the testbed components themselves via virtual sensors and actuators, which mimic some of the physical sensors and actuators of a habitat. Sensors and actuators are controlled via an API that is accessible to external programs such as a GUI or automated control software.

Sensors report on values of the underlying simulation. For example, an O_2 sensor would report the amount of O_2 in the atmosphere. Sensors in the real-world are noisy – that is they do not always return ground truth. Thus, sensors in the simulation are modeled with an adjustable Gaussian noise function. Sensor noise can be turned off so that the sensors report ground truth. Actuators are mirror images of sensors – they allow for control actions to be taken on the simulation. Like sensors, actuators in the real-world are noisy. For example, an injector that is told to open for one second will open for slightly more or less than one second given its mechanical tolerances. This noise is modeled as a Gaussian function. The parameters of the noise function are adjustable and the function can be turned off.

Fault Insertion

Hardware fails and any realistic testbed will need to be able to have faults injected into it. There are many types of faults that could be inserted. Each module can have malfunctions of varying degrees of severity and temporal length. For simplicity, the malfunctions have been divided into two categories based on temporal length: permanent and temporary; and three subcategories of severity: low, medium and high. For example, a temporary but severe malfunction in the potable water store would be a large water leak. A permanent but low severity malfunction in the

power production module would be the loss of a part of a solar array. There is an API for fault insertion into the testbed so that faults can be inserted by any external process (e.g., a GUI). It is also possible to script faults using an XML file that states the kind of fault and the time it should be inserted. When the testbed starts running it consults the script and inserts the fault at the appropriate time. A script allows for repeatable tests to be performed. The API allows for a testbed operator to insert faults in real-time.

Data Logging

An important aspect of any testbed is the logging of all data for later analysis. Currently the two simulations log all simulation values to separate relational databases. This data can then be used for analysis or playback using common database tools. The database to be used is MySQL, which is a free relational database with networking properties. Having separate databases is not ideal and an integrated logging facility will be developed in the future. With distributed simulations the synchronization of data logging is important so the proper cause and effect relationships can be maintained (see (Kortenkamp et al, 2004) for examples).

AUTOMATED CONTROL

The purpose of constructing the testbed is to explore integration and control issues for a lunar outpost. A sustainable outpost will require a significant degree of autonomy from earth-based ground control. A testbed allows us to experiment with different automated control strategies and develop new approaches to closed-loop habitat control. Early tests with life support hardware have shown the effectiveness of automated control in reducing the need for human supervision (Bonasso, Kortenkamp and Thronesbery, 2003; Schreckenghost et al, 2002). An automated control system for a lunar outpost will consist of the following components:

- State determination: Identifies states, state properties, and collections of states and monitors for the occurrence of pre-specified values of states.
- Diagnosis: Identifies system faults and isolates the cause(s) of the fault, when possible.
- Planning: Involves the generation and evaluation of one or more sets of actions (plans) that will move the system under control from an initial state to a desired state.
- Execution: Carries out the tasks specified by a plan.
- Real-time sensing & acting: Directs the hardware actuators to carry out operations.
- Resource management: Administers the allocation, use, and optimization of resources (e.g., power, water, gases, etc.) during a mission.
- Capabilities determination: Identifies the functions available for a given system configuration.
- Mission management: Sets operational objectives, plans to achieve those objectives, and determines the impacts of problems to those plans.

These components all work together to control a habitat. Not all components need to be implemented in software. Some of a component's capabilities might be supplied by a human (e.g., mission management).

EXAMPLE SCENARIO

A simple scenario has been created for use with the testbed and an automated control system. The scenario involves an ECLSS and EPS that are connected, with the EPS providing power needed by the ECLSS. During nominal operations the automated control system manages that ECLSS to ensure appropriate living conditions (e.g., releasing O_2 into the cabins from the O_2 store as needed, releasing nitrogen into the cabins from stores as needed to keep cabin pressure, etc.). At a specified time a failure is inserted into the EPS. The failure causes a switch to trip open stopping power from flowing to the CO_2 removal system. The CO_2 removal system has a short-term battery back-up for power from which it begins to draw automatically. The automated control system is notified of the switch trip. It

begins executing a diagnostic procedure to determine the cause of the trip and to attempt a recovery. In this case, the recovery is to try to reclose the switch, which fails. The automated control system then puts the switch in a safe mode and alerts ground control and crew that a repair will need to be made. The automated control system also begins a controlled shutdown of the CO_2 removal system so it will not have an uncontrolled shutdown when its reserve battery power is depleted. CO_2 levels in the cabin will begin to rise until a repair is completed or a backup system is brought on-line. This simple scenario shows the integration of two simulations, a testbed controller (to insert the failure) and an automated control system. The automated control system contains several of the components listed in the previous section, including state determination, diagnosis, execution, and real-time control.

CONCLUSION

A testbed has been developed that includes an ECLSS and an EPS system with automated control system and evaluations have begun. The testbed is extensible to add additional systems, different outpost configurations, different autonomous control systems and hardware-in-the-loop. The testbed also allows for inserting failures and perturbations to study system robustness and reliability. Future work will involve the use of the testbed to evaluate different mission control scenarios investigating the roles of autonomy, ground control and crew in maintaining a lunar outpost.

ACKNOWLEDGMENTS

Scott Bell of S&K Aerospace was instrumental in implementing BioSim. Kenneth McMurtrie of Tietronix implemented the EPS simulation. All habitat parameters were supplied by Molly Anderson of NASA JSC.

REFERENCES

Bonasso, R. P., Kortenkamp, D., and Thronesbery, C., "Intelligent Control of a Water Recovery System," *AI Magazine*, Vol. **24**, No. 1, Spring 2003.

Finn, C.K., "Dynamic System Modeling of Regenerative Life Support Systems," in Proceedings of the *29th International Conference on Environmental Systems (ICES),* Society of Automotive Engineers (SAE), 1999.

Goudarzi, S. and Ting, K. C., "Top-Level Modeling of Crew Component of ALSS", in Proceedings of the *International Conference on Environmental Systems (ICES),* Society of Automotive Engineers (SAE), 1999.

Kortenkamp, D., and Bell, S., "Simulating Advanced Life Support Systems for Integrated Controls Research," in Proceedings of the *33rd International Conference on Environmental Systems (ICES),* Society of Automotive Engineers (SAE), 2003.

Kortenkamp, D. M., Simmons R., Milam, T., and Fernandez, J. L., "A Suite of Tools for Debugging Distributed Autonomous Systems," in *Formal Methods and Systems Design Journal*, vol. 24, pp. 157-188, 2004.

Linrud, C., Powers, A., Gjestvang, R., MacCallum, T., and Anderson, G., "ECLSS Human-Rating Facility for Testing and Development of New ECLSS Designs," in Proceedings of the *37th International Conference on Environmental Systems (ICES),* Society of Automotive Engineers (SAE), 2007.

Lubman, A. "Cascade Distillation Subsystem Hardware Development for Verification Testing," in Proceedings of the *37th International Conference on Environmental Systems (ICES),* Society of Automotive Engineers (SAE), 2007.

Mas, J.L., "FIPES: Facility for Integrated Planetary Exploration Simulation," in Proceedings of the *37th International Conference on Environmental Systems (ICES),* Society of Automotive Engineers (SAE), 2007.

Miller, A. M., Edeen, M., and Sirko, R. J., "Plant Growth Modeling at the JSC Variable Pressure Growth Chamber: An Application of Experimental Design," in Proceedings of the *International Conference on Environmental Systems (ICES),* Society of Automotive Engineers (SAE), 1992.

Miyajima H., Abe K., Hirosaki T., and Ishikawa Y., "Design of Intelligent Control Software for Mini-Earth," in Proceedings of the *36th International Conference on Environmental Systems (ICES*), Society of Automotive Engineers (SAE), 2006.

Rodriguez, L.F.,.Kang, S., and Ting, K.C., "Top-level Modeling of an ALSS Utilizing Object-oriented Techniques," *Proceedings of the 33rd COSPAR Scientific Assembly*, Manuscript Number F4.3-003, 2000.

Schreckenghost, D., Thronesbery, C., Bonasso, R. P., Kortenkamp, D., and Martin, C. "Intelligent Control of Life Support for Space Missions," *IEEE Intelligent Systems Magazine*, September/October, 2002.

Tri, T.O., "Bioregenerative Planetary Life Support Systems Test Complex (BIO-Plex): Test Mission Objectives and Facility Development," in Proceedings of the *29th International Conference on Environmental Systems (ICES),* Society of Automotive Engineers (SAE), 1999.

Yeh, H.Y., Brown, C., Lin, C., and Ewert, M., "ALSSAT Development Status and its Applications in Trade Studies," in Proceedings of the *34th International Conference on Environmental Systems (ICES),* Society of Automotive Engineers (SAE), 2004.