07ICES-206

Managing Life Support Systems Using Procedures

David Kortenkamp, R. Peter Bonasso and Debra Schreckenghost
TRACLabs Inc at NASA Johnson Space Center, Houston TX 77058

ABSTRACT

International Space Station life support hardware is controlled mainly from the ground by executing standard operating procedures. While some on-board software exists for safety purposes, most commands are sent from ECLSS ground controllers to achieve mission objectives. This will prove unwieldy for extended operations with increasing time delays. This paper presents a new approach to encoding standard operating procedures that provides a path to greater autonomy in life support operations. Software tools will allow for adjustable automation of procedures from either the ground or on-board. The Cascade Distiller System (CDS) being tested at NASA Johnson Space Center is used as an example system.

INTRODUCTION

Flight controllers and crew members rely on standard operating procedures to control life support systems. For space shuttle and space station these procedures are currently authored in Microsoft Word and manually executed using system command and control displays. For station this requires a full-time ECLSS ground controller interacting with an Atmosphere and Consumables Engineer (ACE) in the “back room” for most activities. The procedures are used to calibrate sensors, check smoke alarms, perform oxygen replenishments (called represses), and start up and shut down ECLSS equipment. Procedures are also used to diagnose malfunctions in ECLSS equipment. While most procedures can be performed by either ground controllers or crew members, they are typically performed on the ground when at all possible. This requires sending literally thousands of commands per year from the ground to station ECLSS systems.

We are developing processes and tools that will allow procedures to be authored and executed in an adjustably autonomous fashion. Adjustable automation means that the human may choose at run-time what parts of the procedure should be executed autonomously and what parts should be executed manually. An execution engine manages the interaction between the human and the underlying system. We have also developed a new procedure representation language called PRL that captures the additional information necessary to execute procedures autonomously.

PROCEDURE REPRESENTATION

Procedures are currently represented in natural language on a human-readable display (see Figure 1). They are intended for human consumption not computer consumption. Thus, the representation does not support adjustable automation. Telemetry and commanding information is not encoded in the procedure.

We have been developing a new procedure representation called the Procedure Representation Language (PRL). This language keeps the user-friendly display format of current procedures but augments it with content-based information (e.g., goals, resources, pre-conditions, post-conditions, etc.) required for more autonomous execution. PRL also allows for procedures to be written in a more modular fashion, with larger procedures composed of small procedure “fragments'” that accomplish specific tasks. In this way, new procedures can be easily created and verified as hardware configurations change. Our procedure representation language is an integration of an automated execution language developed at NASA called PLEXIL [1] and the existing International Space Station (ISS) procedure representation. Both of these representations and our new PRL are schemas written in the Extensible Markup Language (XML).

STRUCTURE OF A PROCEDURE

The basic structure of a procedure as represented in PRL is:

Meta data: Includes a unique identifier, the procedure name, the procedure author, date, etc.

Automation data: Includes the following:

- Start conditions: a boolean expression that when evaluated to false means that the procedure

should wait until the boolean expression is true before starting

- Pre-conditions: evaluated after the start condition and if false then exit the procedure immediately with failure
- Post-conditions: evaluated after a procedure is done and if it evaluates to false then the procedure has failed
- End conditions: evaluated continuously and when it is true execution of the procedure is finished
- Invariant conditions: must remain true during the entire execution of the procedure; otherwise the procedure fails
- Resources: any resources (time, fuel, crew members, power, tools, etc.) required for execution of this procedure

Parameters: Declarations of any data that is passed to the procedure from whatever is calling the procedure

Local Variables: Declarations of any variables used internal to the procedure

ExitModes: Definition of explicit procedure exit modes, specifying procedure success or failure, and giving optional description of the reason for exiting

ProcTitle: Procedure number and title

InfoStatement: Specifies explanatory information (e.g. notes, cautions, warnings) that might be needed or desired by a human executor

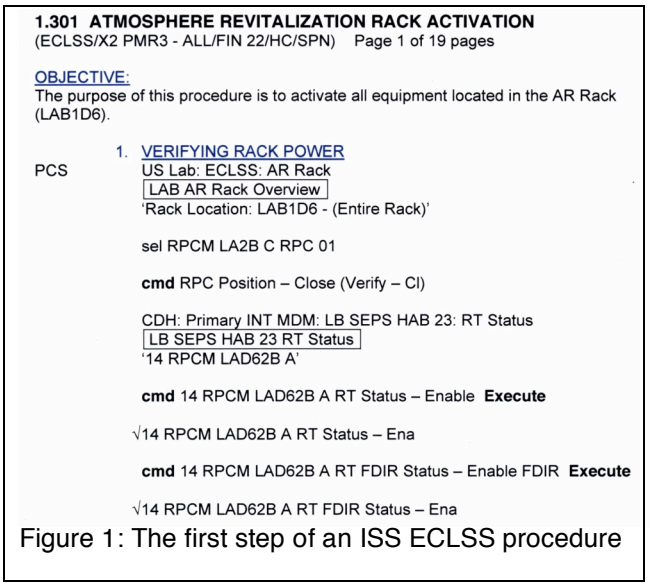
Step: A step is the basic organizing structure of a procedure. A procedure can contain one or more steps and each step consists of the following parts:

- Automation data: As above except replace the word “procedure'” with “step'”
- Step title and unique identifier
- Information to be displayed to the user before this step is executed in manual operations
- A block, which can be ordered (i.e., executed in sequence) or unordered (i.e., executed in any order). A block can also consist of an If-Then statement, a For-Each statement or a While statement. Inside of a block are:
    - Automation data, as above except for blocks
    - Another block allowing for arbitrary nesting of blocks in a step
    - Instructions, which are limited to the following:
        - Command instruction, which sends an electronic command to the system being controlled
        - Ensure instruction, which checks to value of a telemetry variable against a target and, if the value is not correct then issues a command that should make it correct
        - Input instruction, which assigns external data (from a crew member, telemetry, etc.) to a local variable
        - Manual instruction, which asks a human to issue a command
        - Physical device instruction, which asks a crew member to physically manipulate a device
        - Wait instruction, which waits for either a set period of time or until a boolean expression evaluates to true, whichever comes first
        - Procedure call, which can be blocking (i.e., the current procedure pauses until the called procedure finishes) or non-blocking (i.e., the two procedures continue to run in parallel
        - Stop, pause and resume procedure, which effects the execution of the current procedure
- The last component of a step is a conditional branch, which contains a set of boolean expressions paired with a goto step or exit procedure command that is executed if the boolean expression is true. The default is to go to the next step if no conditional branch is given.

This representation captures the content and intent of the procedure along with safety rules (or conditions) under which the procedure and its components are to be executed.

PROCEDURE EXAMPLE

Figure 1 shows the first step of a multi-step ISS ECLSS procedure to activate the atmosphere revitalization



1.301 ATMOSPHERE REVITALIZATION RACK ACTIVATION
(ECLSS/X2 PMR3 - ALL/FIN 22/HC/SPN) Page 1 of 19 pages

OBJECTIVE:
The purpose of this procedure is to activate all equipment located in the AR Rack (LAB1D6).

1. VERIFYING RACK POWER

PCS US Lab: ECLSS: AR Rack
LAB AR Rack Overview
'Rack Location: LAB1D6 - (Entire Rack)'

sel RPCM LA2B C RPC 01

cmd RPC Position – Close (Verify – CI)

CDH: Primary INT MDM: LB SEPS HAB 23: RT Status
LB SEPS HAB 23 RT Status
'14 RPCM LAD62B A'

cmd 14 RPCM LAD62B A RT Status – Enable **Execute**

√14 RPCM LAD62B A RT Status – Ena

cmd 14 RPCM LAD62B A RT FDIR Status – Enable FDIR **Execute**

√14 RPCM LAD62B A RT FDIR Status – Ena

Figure 1: The first step of an ISS ECLSS procedure

system. The procedure is nineteen pages long and is presented to the human or the crew member exactly as shown in Figure 1. In Step 1, which is titled "Verifying Rack Power" the first several lines are instructions on how to navigate to the correct page of the ISS command and control displays. Translated, it tells the person executing the procedure to go to the US Lab page, then the ECLSS page, then the AR Rack page then look for the label "LAB AR Rack Overview" on that page, then find the "Rack Location LAB1D6 – (Entire Rack)" label and select RPCM LA2B C RPC 01. After selecting that the user should see a command button labeled "RPC Position" with the word "Close" in it. They should hit that button and verify that the telemetry talkback reads 'CI'. Assuming the verify is correct they move on to the next instruction in the procedure, which is another navigation through command and control pages and another command. It is easy to see how, over nineteen pages, it can get very tedious to enter commands manually and move your attention between the procedure and the command and control displays.

SYSTEM REPRESENTATION

Procedures describe the processes by which a device or system is operated or debugged. They are oriented towards achieving some task or goal. They do not describe the device or system. However, a representation of the system is necessary for procedure execution. That is, a representation of all of the possible commands, telemetry, states, state transitions, connections and taxonomy of the device or system is required to support procedure authoring and execution. This representation is different from the procedure representation described in the previous section.

COMMANDS AND TELEMETRY

The representation of commands and telemetry is necessary so that the procedure author knows what atomic elements are available to construct a procedure. Furthermore, the executive (manual or automated) must know how to get telemetry from, or send a command to, the controlled system and in what format. Ideally this representation of commands and telemetry should come from the hardware designer or vendor. We have chosen an industry standard representation called XML Telemetric and Command Exchange (XTCE) (http://space.omg.org/xtce/index.htm) for representing commands and telemetry.

We needed to make significant modifications in how we used XTCE to accommodate the authoring and execution of generic procedures. Generic procedures are those in which the actual device being operated is not known until the procedure is executed. In that case, the actual commands and telemetry are not known either. For example, there might be a procedure to calibrate a smoke detector, which works for any smoke detector. When a specific smoke detector is calibrated the commands and telemetry will need to be linked to that device. However, when the procedure is written the commands and telemetry need to be generic for all smoke detectors. We have extended XTCE to include a "type" tag that allows for creating generic classes of devices that all share the same commands and telemetry.

STATES

The representation of states and state transitions is necessary so that the procedure author can reference them in pre-conditions and post-conditions (e.g., don't do this procedure when the device is in this state) and so that the executive can check these states when executing the procedure. The state representation of a device could come early in its design before the hardware implementation and before specification of commands and telemetry. This would allow for early development and testing of procedures against a state model of the device. While we could extend XTCE to add state information, we felt that a separate representation would be more powerful. We have chosen State Chart XML (SCXML) (http://www.w3.org/TR/scxml/) for representing states and state transitions.

TAXONOMY

A system representation also needs to include the components of the system and the relationship between components in some kind of taxonomy. There are two kinds of relationships we expect to capture. First there is a hierarchical relationship between spacecraft components. For example, a spacecraft consists of many systems -- power, life support, navigation, propulsion, etc. Each system has many subsystems and subsystems have components (valves, tanks, switches, etc.). Hierarchy and containment are important to represent in order to allow for efficient display and

reasoning. The second kind of relationship is connectivity between spacecraft components. For example, the output of an oxygen generation system may be connected to an oxygen storage tank. Connectivity is important to represent so that situation awareness displays can be built for humans. We are still in the process of determining appropriate representations -- existing standards such as XML Metadata Interchange (XMI) (http://www.omg.org/technology/documents/formal/xmi.htm) may be appropriate.

PROCEDURE EXECUTION

Procedures can be executed autonomously using an *execution engine*, which interprets the procedure representation and issues commands to the underlying system. There has been a great deal of research in the last decade on procedural execution systems, with some of the more prominent being PRS [2], RAPS [3], and APEX [4]. The Space Station program also has a procedural execution system called Timeliner from Draper Laboratories. While underlying implementation details may change, all procedural executives have similar functions: 1) they have a library of applicable procedures; 2) they choose procedures that are eligible to run by matching start and pre-conditions with system states and telemetry in real-time; 3) they decompose hierarchical procedures into sub-procedures; 4) they dispatch commands to lower level control processes; and 5) they monitor for relevant states in the system.

ADJUSTABLE AUTONOMY

Adjustable autonomy allows a procedure to be executed either by a human, but automation or by both. The goal is to minimize the need for human interaction while maximizing the ability for humans to intervene in procedure execution. Adjustable autonomy must be addressed from the beginning by identifying what parts of a procedure (i.e., steps and instructions) *might* be executed autonomously and what parts *must* be executed by a human. Manual execution is required for parts of the procedure that have no electronic commands. It is also required for parts of the procedure that have no instrumentation to determine if execution was successful. We have identified three basic levels of automation for procedures:

- Manual: The command is dispatched or action is performed by a human with all telemetry verification done by a human
- Automatic: The command is dispatched automatically or the telemetry is verified automatically
- Consent: Command is dispatched automatically, but only after approval by a human

After identifying and storing in the PRL how different parts of a procedure could be executed, the end user interface (see next section) is used to mark whether a step or instruction should be manual, automatic or require consent before execution begins. These assignments must not conflict with the pre-authorized level of autonomy for a procedure step or instruction. As the procedure is executed, the execution engine respects the end user's desires and guides the end user through the various manual and consent actions.

PROCEDURE TRACKING

Procedures will continue to be executed manually in upcoming space missions. There will be actions that can only be done by a person for physical or operational reasons. This poses problems for an adjustably autonomous approach to procedure execution. In a purely automated approach the executive knows exactly what is being done and what the status is. However, if some parts of the procedure are manual, then the current execution status will need to be inferred from telemetry or by direct query to the end user. The procedure tracking process does this inference. It uses all available data to determine which step of the procedure is being executed and what the execution status is. It then makes this available to other processes such as the executive and the end user display.

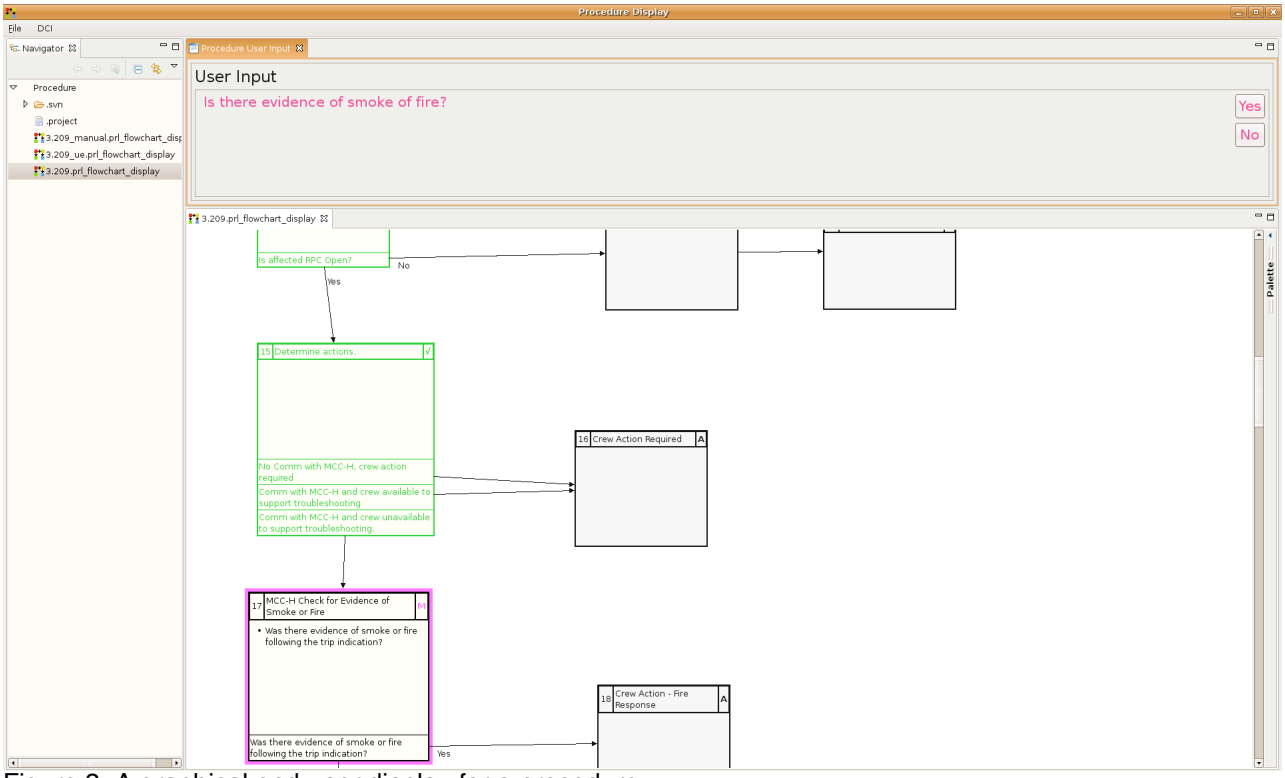


Figure 2: A graphical end-user display for a procedure

GRAPHICAL DISPLAYS AND EDITORS

Humans are an important part of the procedure process. They will be authoring procedures and they will be executing or monitoring the automated execution of procedures.

PROCEDURE DISPLAY

The human executing the procedure will need a display. Ideally, this display will look similar to the ``paper'' procedures of today, but allow for direct execution of commands and direct display of telemetry by the end user. The end user interface will also have to display the results of procedure tracking by highlighting or otherwise noting steps that have been completed, steps that are currently executing, and steps that are pending. Execution status, especially failures, will need to be made explicit. Support for adjustable autonomy requires an easy way to note steps that should be done manually and steps that should be done automatically.

For a single procedure being done by a single end user the display requirements are not demanding. However, if we begin to address multiple procedures being executed in parallel by one or more end users with interaction between them, then a wider variety of issues arise. These include notification of the status of other end users or executives, interruption reasoning, and multi-modal notification. For example, a procedure may have a step that is a wait for some lengthy period of time (say one hour) at which point an end user could begin another parallel procedure. At the end of the hour the end user would need to be notified appropriately, find a break-point in their new procedure, return to the old procedure, get situated, and begin executing. In another example, two end users that are separated by distance may be performing a single procedure (say an EVA astronaut and a ground controller) and need to coordinate their activities. We have begun addressing these issues in a separate project [5].

Figure 2 shows an end user display we developed for ISS procedures. Each box represents a step of the procedure. Steps can branch to other steps based on their outcome. Different colors represent the status and level of autonomy of each step. Green steps are complete. Purple is the current, manual step. At the top messages for the user are displayed and user input is requested.

PROCEDURE EDITING

Procedures will need to be authored, viewed, verified, validated and managed by a variety of people, many of who will not understand XML or other representations. We are developing a Procedure Integrated Development Environment (PRIDE) that will provide an integrated set of tools for dealing with procedures. These tools will allow for authoring, verifying, validating and managing procedures. Authors will be able to graphically design their procedure using palettes of commands, telemetry and procedure constructs. Access to desktop system simulations will be integrated with the PRIDE tool for verification and validation activities. A workflow system will also be integrated with PRIDE to track procedure

changes and approvals. The PRIDE tool will allow authors to view the procedure as it will appear to the end user.

CASE STUDY

NASA Johnson Space Center will be conducting tests in the spring and summer of 2007 of a new water recovery system being built by Honeywell. The new system is called the Cascade Distiller System (CDS). We are developing procedures for controlling the CDS using the tools described in this paper. We have expressed the telemetry and commands of the CDS in an XTCE file. We have written several CDS procedures in PRL and also encoded them in an execution engine called RAPS. The execution engine will execute the procedures against the CDS hardware in an adjustably autonomous fashion. PRL allows for easy expression of the basic CDS operating paradigm. For example, here is a pseudo-PRL snippet for starting up the CDS (the real PRL in XML is quite verbose and difficult to understand without proper editing tools):

Start Procedure CDS Startup

- **Pre-conditions**: coolant flowing AND vacuum pressure nominal AND feed tank full

- **Step 1: Start Systems**
  1. Command Instruction: Start distiller water flowing at 1200RPM
  2. Execute Procedure: Apply Vacuum To All Systems
  3. Execute Procedure: Fill Cold Loop With DI Water
  4. Execute Procedure: Fill Hot Loop With Feed
  5. Execute Procedure: Start Product Flowing
- **Step 2: Wait for System to Start**
  1. Wait for: Product tank weight increase

End Procedure CDS Startup

The pre-conditions ensure that the system is in the proper state for startup. The first step has a command instruction that is issued directly to the hardware. The other four instructions call additional procedures that do various parts of the job and return to this procedure when they are done. For example, the third instruction calls a procedure to fill the cold loop of the CDS with de-ionized water. This is done with a procedure that turns a series of valves. Here is some pseudo-PRL for that procedure:

Start Procedure Fill Cold Loop With DI

- *Pre-conditions*: distiller speed EQ 1200 RPM
- *Timeout*: 60 seconds

- **Step 1: Start DI Water Flowing**

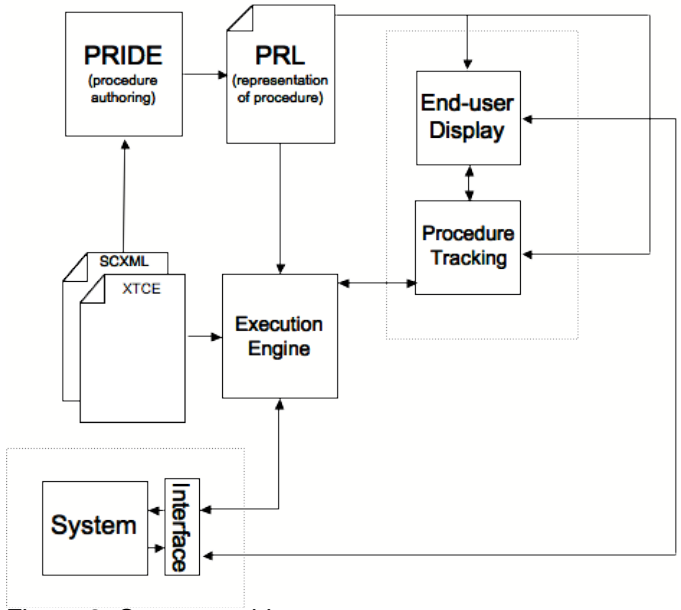


Figure 3: System architecture

1. Execute Procedure: Apply vacuum to distiller system
2. Command Instruction: Open valve to product tank
3. Command Instruction: Open DI water valve
4. Wait for: Cold loop filled
5. Command Instruction: Close DI water valve

End Procedure Fill Cold Loop With DI

This procedure has only one step. It has a precondition that that distiller system be running with a speed of 1200 RPM. The timeout states that this procedure should finish executing in 60 seconds. If it doesn't the procedure aborts with a failure. The first instruction calls another procedure that applies vacuum to the distiller system. The next two steps open valves. The fourth step waits for a signal from the hardware system that the cold loop is filled. The final step closes the DI valve.

We have identified several dozen procedures that will control the functioning of the CDS. These procedures connect via commands and sensors to the underlying CDS via a hardware interface layer as defined in the XTCE representation. Execution is done mainly autonomously using the execution engine, but the level of autonomy for most steps can be set to manual if desired. For example, in the previous procedure if Step 1 is set to manual then a human would be responsible for executing the vacuum procedure and opening and closing all valves. The execution engine would still verify pre-conditions to ensure safe operation.

In addition to our work with the CDS we have also represented several ISS Electrical Power System (EPS) procedures in PRL. We have then executed those procedures against a high-fidelity simulation of the ISS (called ISS-in-a-box) to validate that our approach can

help enhance space station operations. Figure 3 shows the basic system architecture. The system in the lower right can be the CDS or the ISS (or ISS simulation). The interface connects the execution engine to the system. The execution engine executes procedures, which are authored in PRIDE. The procedure tracker and end-user display take data from the execution engine and the system and show the user the current procedure status and help the user perform manual operations or allow for consent to be given.

CONCLUSIONS AND FUTURE WORK

Procedures are the means by which any spacecraft system, including life support systems, are operated. Current operations are predominantly manual and are time intensive and error-prone. We are using simulations and ground testbeds to validate more automated approaches to controlling spacecraft systems. These approaches will be necessary as we develop lunar outposts.

The CDS system will be operated at NASA JSC during the spring and summer of 2007. We will validate our control approaches using this system. We are also re-authoring several existing ISS ECLSS procedures in PRL and will experiment with adjustable autonomous execution against a high-fidelity ISS simulation.

ACKNOWLEDGMENTS

This work was conducted under NASA's Exploration Technology Development Program's Spacecraft Autonomy for Vehicles and Habitats project. The authors wish to thank their colleagues at NASA Ames Research Center who participated in discussions underlyng many of the ideas presented in this paper, including Ari Jonsson, Vandi Verma and Michael Dalal. The authors also wish to thank S&K Aerospace employees Scott Bell, Kevin Kusy, Tod Milam, Arthur Molin and Mary Beth Hudson who work at NASA JSC and implemented many of the software processes described in this paper. The authors also acknowledge Lui Wang of NASA JSC and Robert Phillips of L3Com at NASA JSC for helping create the Procedure Representation Language.

REFERENCES

1. Vandi Verma, Ari Jonsson, Corina Pasareanu, Reid Simmons and Kan Tso, "Plan Execution Interchange Language (PLEXIL) for Executable Plans and Command Sequences," in *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation (i-SAIRAS),* 2005.
2. Michael P. Georgeff and Francois Felix Ingrand, "Decision-making in an Embedded Reasoning System," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1989.
3. R. James Firby, "An Investigation into Reactive Planning in Complex Domains," in *Proceedings of the National Conference on Artificial Intelligence*, 1987.
4. Michael Freed, "Managing Multiple Tasks in Complex, Dynamic Environments," in *Proceedings of the National Conference on Artificial Intelligence*, 1998.
5. Debra Schreckenghost, Carroll Thronesbery, R. Peter Bonasso, David Kortenkamp and Cheryl Martin, "Intelligent Control of Life Support for Space Missions," *IEEE Intelligent Systems*, 17(5), 2002.

CONTACT

The authors may be contacted through email at korten@traclabs.com, bonasso@traclabs.com and ghost@ieee.org