Adjustably Automated Ground Control Procedure Execution for ECLSS Systems

David Kortenkamp
Scott Bell
Debra Schreckenghost
TRACLabs Inc.
{korten,scott,schreck}@traclabs.com

Procedures are used to control all aspects of the International Space Station (ISS) including the Environmental Control and Life Support Systems (ECLSS). Ground controllers use procedures on a daily basis. Current procedures are authored by flight controllers in Microsoft Word and displayed either in PDF or in a specialized XML viewer. There is no link between procedures and system data. In this paper, we present an integrated tool suite for authoring and performing procedures that links procedures directly to system data. The integrated tool suite also offers a path to automation of routine procedures.

I. Introduction

Procedures are routinely used by ground control personnel to monitor and control Environmental Control and Life Support Systems (ECLSS) on-board the International Space Station (ISS). Procedures represent the combined knowledge of systems experts as to the correct operation of complicated systems. The current ISS procedure process relies on Microsoft Word for authoring and either PDF for display of procedures or re-authoring procedures into XML for display in a special-purpose viewer. This is cumbersome and error-prone and does not take advantage of recent advances in human-computer interaction and automation. In this paper, we present an alternative procedure authoring and execution process and present preliminary results.

Our tools consist of an electronic Procedure Representation Language (PRL), a drag-and-drop editing suite for PRL called the Procedure Integrated Development Environment (PRIDE), and a procedure viewer and executor called the Procedure Agent for Execution (PAX). These technologies are currently currently being used for a subset of ISS procedures. We have encoded an existing ISS ECLSS procedure in PRL using PRIDE and executed it against a high-fidelity ECLSS simulation using PAX. Results from this will be presented along with future work and an examination of the roadblocks in moving NASA to electronic, adjustably autonomous procedures.

II. Approach

Our approach to procedures is an integrated tool suite built around an XML-based representation. We first describe the representation and then the tools that produce and use this representation.

A. Procedure representation language

PRL is an XML schema that defines a variety of tags that can be used to describe a procedure.[7,8] There is also an Eclipse-based, drag-and-drop editing environment to create PRL procedures.[5] In PRL, the highest level is a procedure tag that marks the beginning of a new procedure. Each procedure consists of steps that describe smaller tasks within the procedure. Steps themselves have blocks that are containers for instructions that provide explicit detail about commanding a system. Each of these components can have automation data that controls their execution status.

1. *Procedures*

A procedure is the top-level entity in PRL. Each procedure has a human-entered name and number. Each procedure also has a unique identifier. A procedure can contain a block of "meta-data" with information about the procedure such as the author, comments, revisions, etc. Each procedure can contain parameters that are passed into the procedure at execution time. A procedure can also contain local variables that can be used within the procedure. All procedures can have Automation Data that controls when and how they are executed. A procedure has as its body one or more steps.

2. *Steps*

A step has a specific purpose or goal within the procedure. Each step has a human-entered name, a number that is generated sequentially and a unique identifier. Each step has an optional information statement, which is human-readable text that can provide additional information to a human performing the step. Each step must end in one of three ways: 1) A conditional branch in which Boolean expressions are paired with step identifiers and execution branches to the first step whose Boolean expression evaluates to TRUE; 2) A goto-step in which execution continues at the step identified in the goto-step; and 3) An exit procedure in which execution ends. Each step can have Automation Data controlling its execution. Each step has as its body one or more blocks.

3. *Blocks*

Blocks are wrappers that contain the instructions necessary to accomplish the step. The most basic block is an Ordered Block, which contains one or more instructions that are executed one after the other. An Unordered Block contains one or more instructions that can be executed in any order. Other block constructs offer control over execution flow such as if-then, repeat-until and while. Each block contains another block or a group of instructions.

4. *Instructions*

Instructions are the atomic actions of PRL. There are a wide variety of instructions, often tailored for different disciplines. A Command Instruction issues a computer command (possibly with parameters) to the underlying system. A Verify Instruction compares a specific telemetry item to a target value. If the comparison is TRUE the instruction succeeds and execution continues. If the comparison is FALSE then execution halts and the procedure fails. A Wait Instruction halts execution either for a specified period of time or until a Boolean expression becomes TRUE. The Call Procedure Instruction calls another procedure using the procedure identifier and passes any required parameters. Those are just a few of the more important instructions in PRL. Each instruction can also have Automation Data that controls its execution.

B. Procedure authoring

While PRL is a good representation for procedures, it is not meant to be authored directly. TRACLabs, along with another NASA contractor, has developed a drag-and-drop authoring environment called the Procedure Integrated Development Environment (PRIDE).[5] Figure 1 shows a screen shot of PRIDE being used to author an ECLSS procedure. The left pane of PRIDE contains all of the procedures on which the author is working. The middle pane of PRIDE is the canvas on which the author can drag procedure elements to create the procedure. At the bottom of the middle pane is a properties window in which the different elements can be customized. The right, skinny pane is a palate of procedure elements that can be dropped on the canvas. The far right pane is a view of a database of commands and telemetry for different systems. Dragging-and-dropping a command or telemetry from that pane onto the canvas automatically creates a command or verify instruction respectively with the appropriate system information. This is what allows PRL to understand system commands and telemetry.

For the work described in this paper, the database of commands and telemetry for the ECLSS system were described in an Ontology Web Language (OWL) file. The PRL references this OWL file when describing which command to execute or which telemetry item to monitor. For a given command, the OWL file contains information regarding the ID of the command (called a Program Unique Identifier or PUI for ISS), the operational nomenclature, the command arguments, their data types, and what system component. For a given telemetry, the OWL file contains the ID of the telemetry (PUI), the operation nomenclature, the data type of the telemetry, and to what system component it belongs.

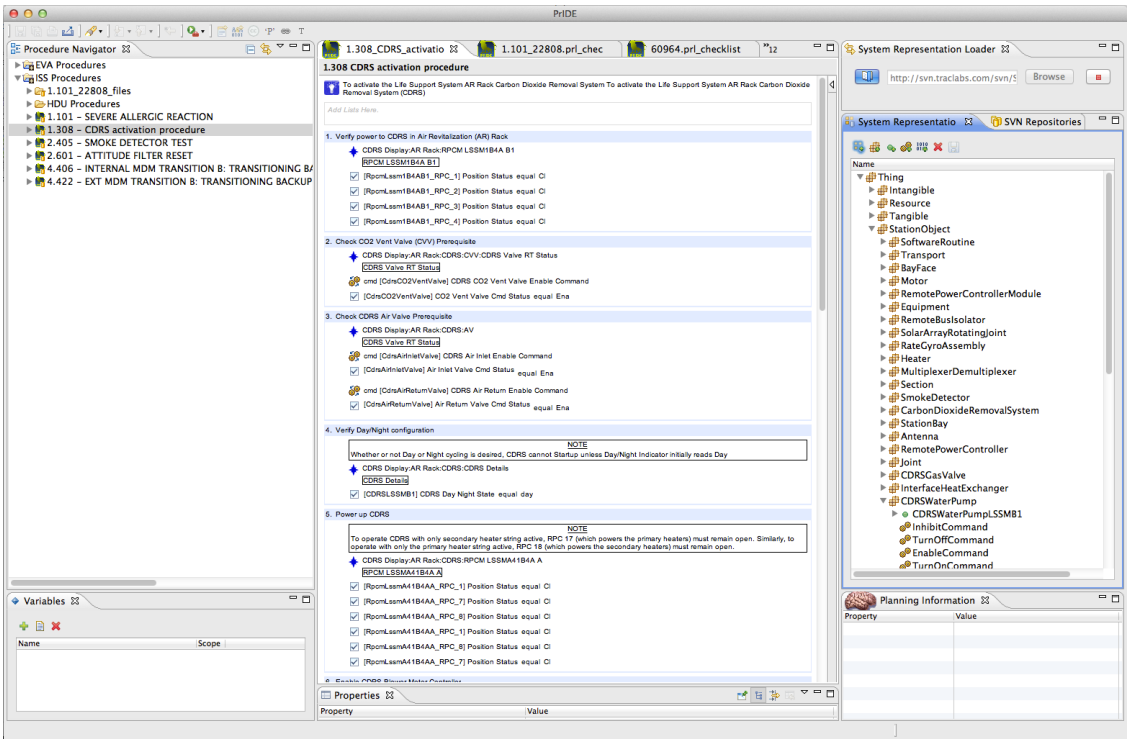

Figure 1. The PRIDE procedure authoring environment.

C. Procedure viewing

1.308 NODE 3 CDRA ACTIVATION
(ECLSS/X2R9 - ALL/FIN 4/HC/SPN) Page 1 of 18 pages

OBJECTIVE:
To activate the Node 3 AR Rack Carbon Dioxide Removal Assembly (CDRA).

NOTE
Steps 1 to 3 assume that the Node 3 AR Rack has been powered and ventilation/smoke detection activated per 1.307 NODE 3 ATMOSPHERE REVITALIZATION RACK POWER AND VENTILATION ACTIVATION (SODF: ECLSS: ACTIVATION AND CHECKOUT: ARS).

1. VERIFYING RACK POWER

PCS
Node 3: ECLSS: AR Rack
Node 3 AR Rack Overview
'Rack Location: NOD3A4 - (Entire Rack)'

✓RPCM N31B4A B1 RPC 04 RPC Position – Closed

CDH: Primary HCZ MDM: LB SEPS N3 14: RT Status
LB SEPS N3 14 RT Status
'15 RPCM N3A41B4A A'

✓15 RPCM N3A41B4A A RT Status – Ena
✓15 RPCM N3A41B4A A RT FDIR Status – Ena

2. VERIFYING SMOKE DETECTOR ACTIVE
Node 3: ECLSS: AR Rack
Node 3 AR Rack Overview
'SD'

✓RPCM N3A41B4A A RPC 02 RPC Position – Closed
✓Monitoring Status – Enabled

Figure 2. A snippet of an ECLSS procedure as it is currently viewed by flight controllers.

The PRL must be displayed to the ground controller in order for them to perform the procedure correctly. Currently, procedures are displayed either using PDF or a specialized tool called the International Procedure Viewer (IPV). Neither PDF nor IPV allow procedures to be performed with embedded telemetry and commanding displays. Figure 2 shows the beginning of an ECLSS Node 3 Carbon Dioxide Reduction Assembly (CDRA) activation procedure. The

first few lines after the step title provide a navigation path so that flight controllers can find the right display. Then the line with a check in front of it tells the flight controller to verify that the position of a Remote Power Controller (RPC) is Closed. The flight controller needs to use the navigation paths to find that piece of telemetry on their data screens. This process is inefficient and error-prone.

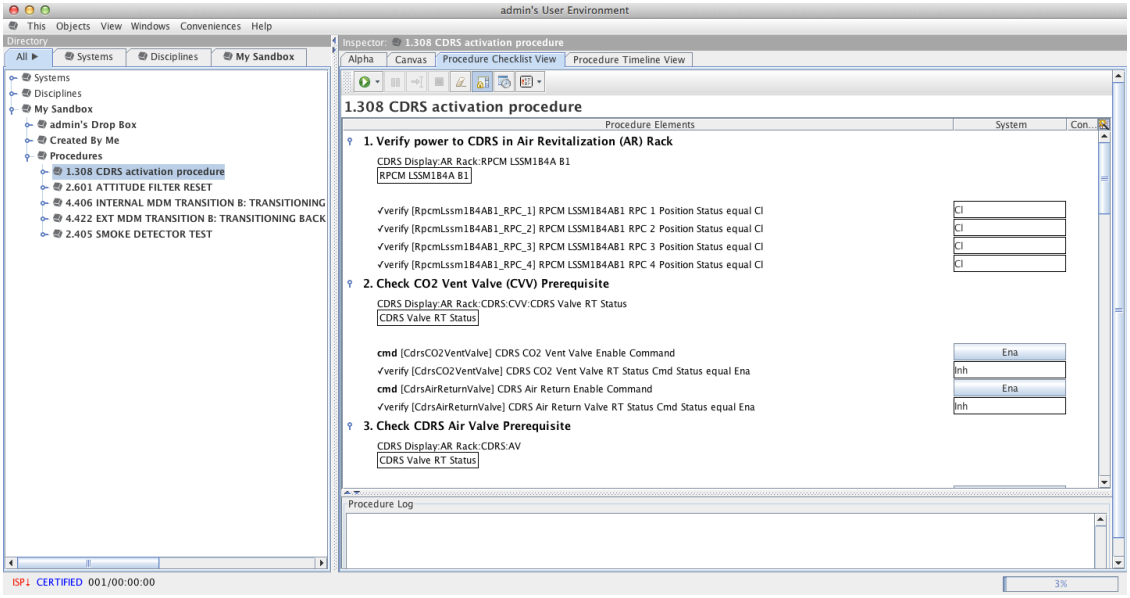

Figure 3. A procedure viewer that integrates telemetry and commands.

We have developed a procedure viewer that reads in PRL, connects to the system telemetry and commands, and displays an integrated procedure view to the flight controller. Figure 3 shows the new, integrated viewer. The navigation paths are still right under the step title in case flight controllers want to double check the data. However, immediately to the right of the verify instruction is the actual live telemetry coming from the system. This allows the flight controller to verify the telemetry immediately. This can be done because the PRL file contains the necessary information to connect the procedure viewer to the telemetry stream.

D. Procedure automation

One the PRL contains the telemetry and commands necessary to perform the procedure, those commands can be sent by the computer and the telemetry compared to target values also by the computer. In this way, procedures can be performed automatically under flight controller supervision. We have developed a computer program called the Procedure Assistant for Execution (PAX) that reads the PRL file and issues commands and checks telemetry. PAX interacts with the procedure viewer described above to show flight controllers the current automation state as well as allowing flight controllers to supervise PAX. Figure 5 shows the procedure viewer connected to PAX. The green instructions have already been evaluated. The blue line is the current point of automated execution.

Breakpoints can be inserted into the procedure at any location. Breakpoints tell PAX not to proceed with autonomous execution until an operator gives consent. Breakpoints are created by simply right-clicking on a procedure element and selecting a breakpoint. They appear as small stop signs in the procedure display (see Figure **??**). Note that setting a breakpoint at every instruction in the procedure is not equivalent to manual execution. Even with breakpoints, PAX is still verifying telemetry and issuing commands, which is not the case in manual mode. Breakpoints do allow operators to ensure that the context for the procedure is still valid.

Sometimes a part of a procedure may not be relevant in the current context. Rewriting the procedure to match the context is too time consuming, so in current operations a *flight note* is attached to the procedure by a flight controller to modify it temporarily. Typically, flight notes instruct the operator to skip specific parts of the procedure. To replicate this situation, we allow the operator to mark specific instructions in the procedure as ones to be skipped (see Figure 6). PAX will then not execute those in automated mode. These notations do not affect the underlying PRL, which remains intact.

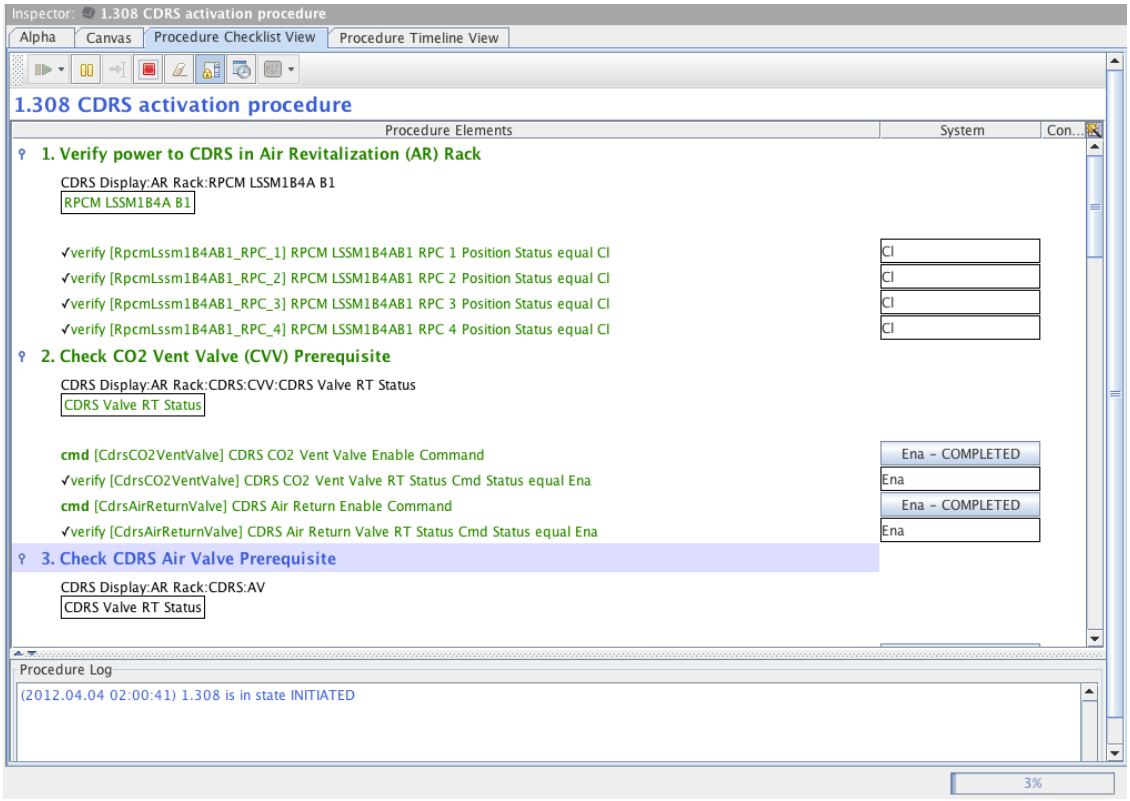

Figure 4. Automated execution of a ECLSS procedure.

Skipping is the main, run-time alteration of a procedure. Several shorthand operator controls have been developed to provide easier control over a procedure. For example, PAX can be told to start from a specific point in the procedure (equivalent to skipping the prior instructions). PAX can also be told to execute only parts of the procedure (equivalent to skipping all the rest). PAX can also be told to restart the procedure at a specific point after it has been stopped, paused or PAX detects a failure condition. This might be used, for example, to resend a command that has failed. PAX can be told that specific instructions are already completed (identical in functionality to skipping them, but rendered differently on the display). Each of these capabilities allows an operator to tailor procedure execution using PAX to their current mission needs.

PAX detects off-nominal situations that are described in the PRL, halts execution, and marks the problematic instruction and the procedure as failed (see Figure 7). Off-nominal situations include commands that fail to send, out of bounds telemetry values, etc. Also note in this figure that all actions taken by PAX are logged. The log can be seen at the bottom of the window. PAX publishes all state information of procedure execution using JAX-RS, a Java implementation of Representational State Transfer[3] or REST. The RESTful interface is essentially a web service running on the same machine as PAX. The following are valid data requests to PAX:

- `/available` - lists available procedure PAX can execute
- `/available/(id)` - returns the PRL of the specified procedure
- `/procedures` - lists currently executing procedures in PAX along with their status and runtime ID
- `/procedures(runtime_id)` - lists execution log of specified procedure. For example, when the procedure started, when a particular instruction finished, etc.

The above requests can also be modified with date, status, and result filters. Valid statuses for procedures are UNKNOWN, INITIATED, FINISHED, PAUSED, STOPPED. Valid results for a procedure are UNKNOWN, SUCCEEDED, FAILED, ABORTED. The following are valid requests to PAX for it to do something:

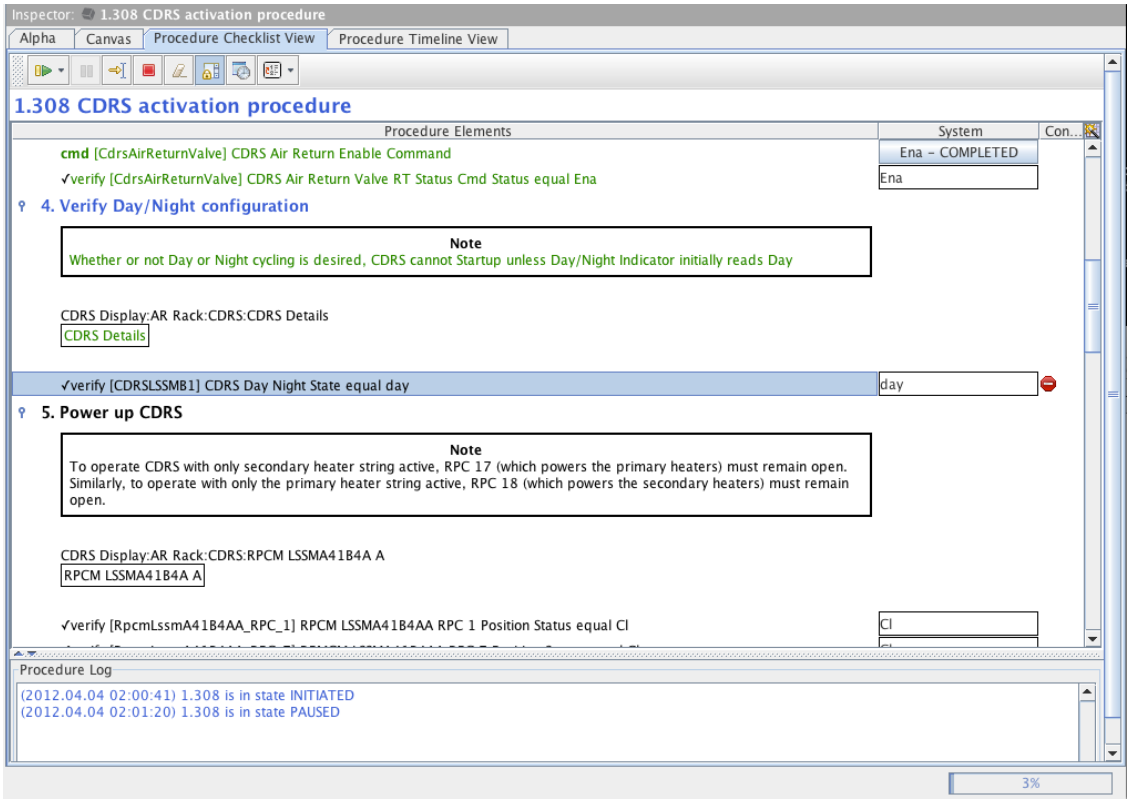

Figure 5. Inserting a breakpoint for automation into an ECLSS procedure.

- `/available/(id)` - starts execution of a procedure and returns the new runtime ID

We are currently using this remote access to generate reports of procedure execution metrics, e.g., how long the procedure took to execute, if it was executed correctly, and who executed it. We also plan to use remote access to drive external web sites and smart phone apps to monitor procedure execution.

III. Results

An actual ISS CDRS activation procedure was authoring in PRIDE and PRL. A subset of that procedure was used to control a simulated ISS CDRS. The simulated CDRS was implemented using the BioSim environment, which is a high-fidelity life support simulation developed by TRACLabs.[6] A screen shot of the simulation is shown in Figure 8. The simulation allowed us to send commands to the CDRS and get appropriate telemetry in response. It also allowed us to inject failures to test PAX's error detection functions. We also implemented a full set of command and control displays in BioSim for the CDRS (also shown in Figure 8). This allowed us to perform the procedure in the current fashion, without integrated commands and telemetry and to perform the procedure with our integrated procedure viewer. The test procedure had fifteen steps. This took approximately 6.5 minutes to perform using the current viewer and CDRS command and control displays. The same procedure takes approximately 3 minutes using the new, integrated procedure viewer described in this paper. The automated procedure does not run any faster because we set the automation speed slow enough to allow for human supervision and intervention if necessary.

The PRIDE authoring tools was also evaluated using a set of medical operations procedures.[4] This showed that the overall time required to author a procedure using Word and then having that Word file converted to the IPV XML was approximately three days. This was mostly due to the process of transferring the Word file to an XML expert, who then re-typed that document into XML and sent it back to the original author for verification. PRIDE can produce IPV XML directly and authoring time was less than one day.

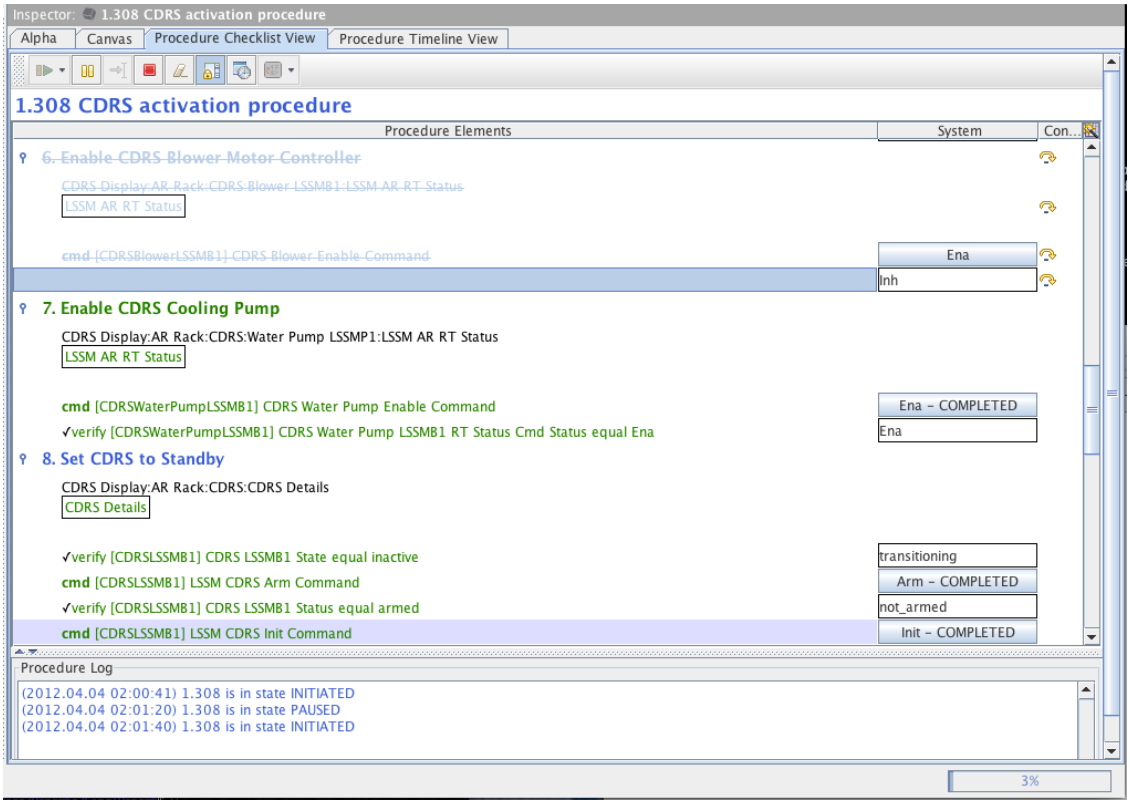

Figure 6. Skipping a step in an ECLSS procedure.

IV. Future Work

Treating procedures as information represented in PRL opens up a large number of new research directions. Activity plans can be built automatically using information contained in PRL (see[1] for an early version of this). We are exploring using plan recognition techniques, including video analysis, to detect when procedures have been manually started so as to alert ground controllers to crew activities. Previous work on plan recognition for procedure execution was not linked to PRL, but showed the usefulness of the approach.[2] Efforts are also underway to use procedure performance metrics to monitor crew fatigue and stress. Some complex procedures are performed by several operators working in parallel on multiple subsystems. Our procedure displays can be extended to coordinate these activities and maintain awareness across the team. We are also looking to explore the use of procedures in robot operations and have performed some preliminary investigations.[9] We are working to integrate auxiliary system displays with automated procedure execution for further situational awareness. Finally, we are exploring the use of electronic procedures on long-duration missions with significant time delays. In these cases, flight controllers cannot execute procedure remotely. The crew will need to have on-board procedure execution and assistance. This may include videos, automation, and other support infrastructure.

V. Conclusion

Procedure authoring, verification, and performance are amongst the most costly and time-consuming activities with respect to mission operations for life support. However, having accurate and up-to-date procedures that can be performed in an less error-prone and timely fashion is vital to the safe operation of life support systems in space. Our suite of tools, PRL, PRIDE, procedure viewer, and PAX, provides the foundation upon which electronic procedures can be applied to a variety of space systems.

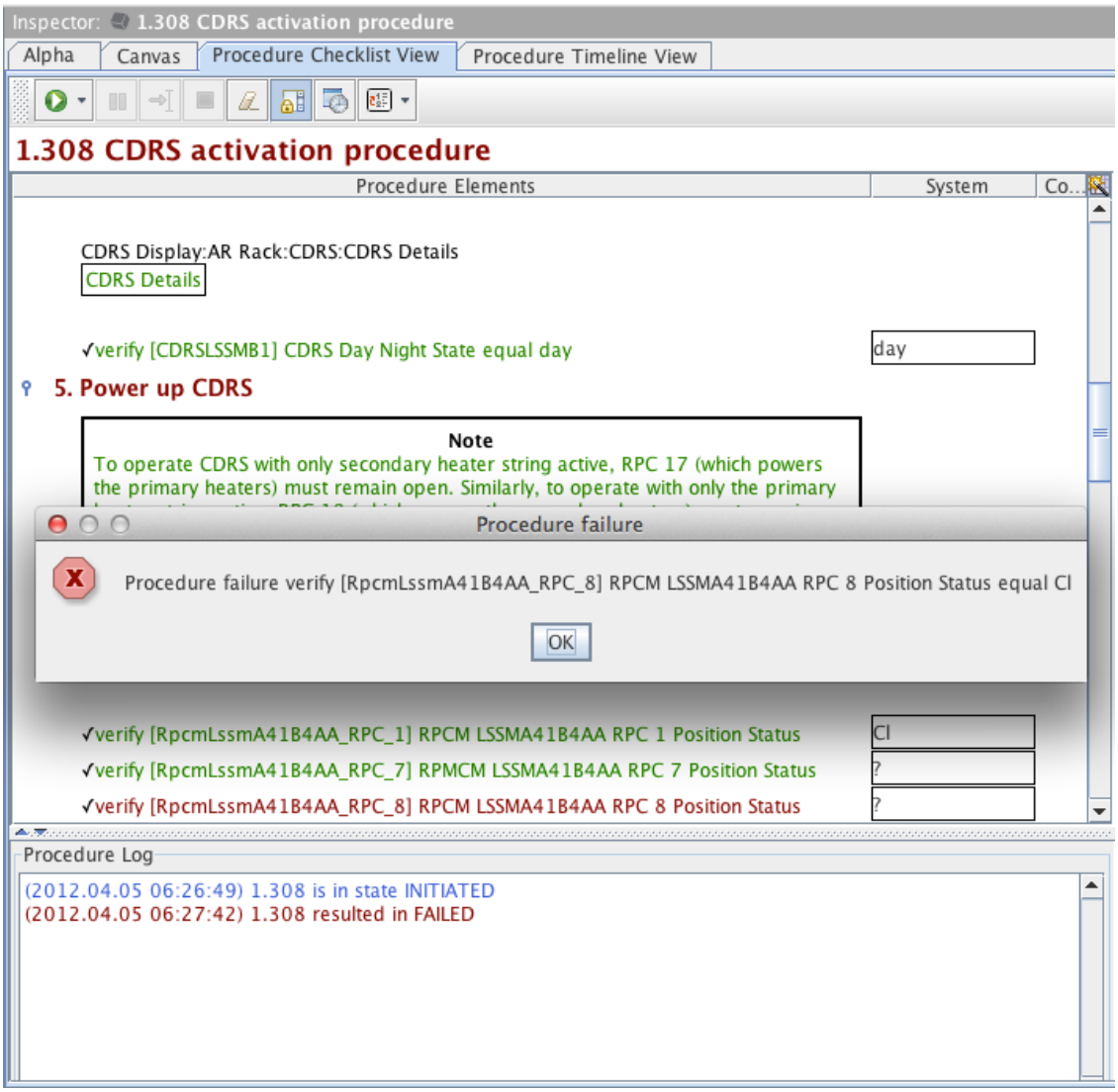

Figure 7. Telemetry failing to match in an ECLSS procedure.

VI. Acknowledgements

This work is funded under NASA contract NNX10CA18C. The authors wish to thank Jeremy Frank of NASA Ames Research Center and Alan Crocker of NASA Johnson Space Center for their contributions to the ideas presented in this paper.

References

[1]Mark Boddy and R. Peter Bonasso. Planning for human execution of procedures using anml. In *Proceedings of the International Workshop on Planning and Scheduling for Space (IWPSS 2009)*, 2009.

[2]R. Peter Bonasso, David Kortenkamp, and Troy Whitney. Using a robot control architecture to automate space shuttle operations. In *Proceedings of the 1997 Innovative Applications of Artificial Intelligence Conference*, 1997.

[3]Roy Thomas Fielding. Architectural styles and the design of network-based software architectures, 2000.

[4]Mary Beth Hudson, Arthur Molin, and David Kortenkamp. Electronic procedures for medical operations in space. In *Proceedings of the AIAA Space 2011 Conference and Exposition*, 2011.

[5]Michel Izygon, David Kortenkamp, and Arthur Molin. A procedure integrated development environment for future spacecraft and habitats. In *Proceedings of the Space Technology and Applications International Forum (STAIF 2008), available as American Institute of Physics Conference Proceedings Volume 969*, 2008.

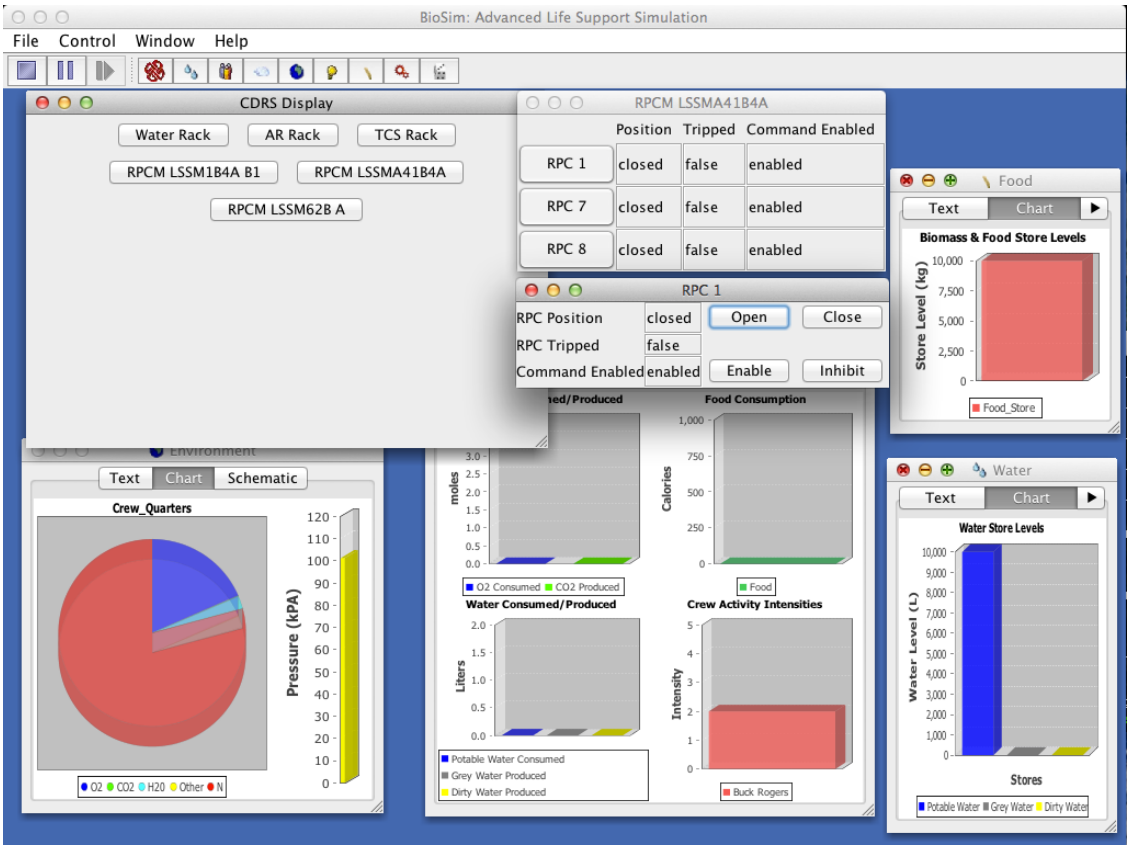

Figure 8. A simulation of the CDRS complete with command and control displays.

[6]David Kortenkamp and Scott Bell. Simulating advanced life support systems for integrated controls research. In *Proceedings International Conference on Environmental Systems*, 2003.

[7]David Kortenkamp, R. Peter Bonasso, and Debra Schreckenghost. Developing and executing goal-based, adjustably autonomous procedures. In *Proceedings AIAA InfoSysAerospace Conference*, 2007.

[8]David Kortenkamp, R. Peter Bonasso, and Debra Schreckenghost. A procedure representation language for human spaceflight operations. In *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-08)*, 2008.

[9]Debra Schreckenghost, Tam Ngo, Robert Burridge, Lui Wang, and Michel Izygon. Remote task-level commanding of centaur over time delay. In *Proceedings of the Space Technology and Applications International Forum (STAIF) (AIP Conference Proceedings Volume 969)*, 2008.