Embedding Procedure Assistance into Mission Control Tools

Scott Bell and **David Kortenkamp**
TRACLabs Inc.
Houston, TX
scott@traclabs.com, korten@traclabs.com

Abstract

Procedures are pervasive in spacecraft operations and are used to control every aspect of human spaceflight. Both ground controllers and crew members use procedures on a daily basis. Current procedure operations are document-focused with little computer support. This paper describes an information-focused procedure representation and the development of procedure assistance tools that allow for adjustably autonomous procedure operations. The procedure representation is XML-based with direct links to system sensors and actuators. The procedure assistant can use the XML representation to execute autonomously a procedure under careful human supervision. This approach allows current operations to continue while providing a path to increased automation over time and as the situation allows.

1 Introduction

Procedures are the accepted means of commanding spacecraft. They encode the operational knowledge of a system as derived from system experts, testing, training and experience. NASA has tens of thousands of procedures for Space Shuttle and the International Space Station (ISS), which are used daily by both flight controllers and crew. In current Space Shuttle and ISS operations, procedures are stored as Word files or in display-oriented XML schemas. That is, they are treated as documents and maintained as such. Procedures are displayed to operators using applications separate from the applications used to display commands and telemetry. This means that procedures cannot currently interact with commands and telemetry to help an operator's situation awareness. As NASA begins to adopt new commanding and display technologies under the Mission Control Technologies (MCT) project, there is an opportunity to rethink procedures and their relationships with command and display interfaces. The goal is to treat procedures as information that can be manipulated by computer programs that are integrated with the spacecraft's command and data handling systems.

In current operations, procedures are executed manually using pre-defined command and control displays. Manual execution of procedures can be useful in complex situations or when the state of the spacecraft is unknown or uncertain. However, it is time consuming in nominal situations and doesn't allow for monitoring procedure execution for safety and training purposes. We are developing an adjustably autonomous procedure assistant that can autonomously execute parts or all of a procedure by checking telemetry and issuing commands while also allowing for manual execution.

2 Background

Two key technologies being developed at NASA have created an opportunity to significantly change the manner in which procedures are managed and executed by flight controllers. These two technologies are the Procedure Representation Language (PRL) and new mission control telemetry and commanding displays being developed by MCT. We discuss each of these briefly.

2.1 Procedure Representation Language

PRL is an XML schema that defines a variety of tags that can be used to describe a procedure [Kortenkamp *et al.*, 2007; 2008]. There is also an Eclipse-based, drag-and-drop editing environment to create PRL procedures [Izygon *et al.*, 2008]. 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.

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.

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.

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.

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.

Automation Data
Automation Data is used to control execution in PRL. Automation Data includes a pair of gating conditions called PreConditions and PostConditions. These are Boolean expressions that must evaluate to TRUE before execution of the procedure, step or instruction begins and after execution ends or the procedure execution fails. Automation Data also includes a pair of wait conditions called StartConditions and EndConditions. These are also Boolean expressions that must evaluate to TRUE before execution can begin and end. If these conditions evaluate to FALSE then execution waits until they become TRUE. Automation Data also includes InvariantConditions that are Boolean expressions that must remain TRUE throughout execution of the procedure, step or instruction or execution fails. Automation Data also includes a description of the resources necessary to execute the procedure, step or instruction. Automation Data can apply at the procedure, step or instruction level.

2.2 Mission Control Technologies

The Mission Control Technologies (MCT) Project at NASA Ames Research Center is developing technologies to change fundamentally the way applications for mission control are designed, constructed and deployed [Trimble *et al.*, 2006]. Rather than building software as monolithic applications, MCT enables software to be built from fine-grained, end-user composable components and services from which software functionality may be assembled and easily modified. The current focus of the MCT project is on telemetry and commanding displays. The core concepts of MCT are: user objects, manifestations of user objects, user composable groups, distributed object sharing and a consistent visual style [Group, 2009].

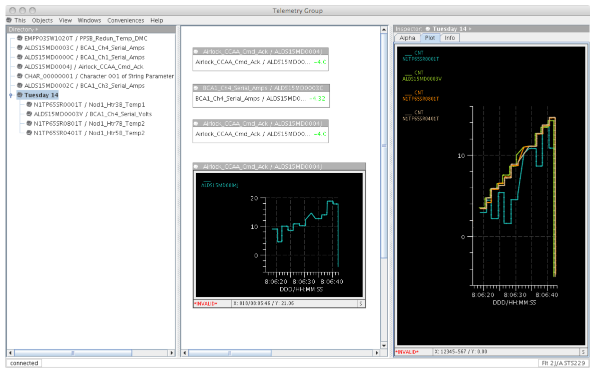

Telemetry Group
This Objects View Windows Conveniences Help
Directory
EMPP03SW1020T / PPSB_Redun_Temp_DMC
ALDS15MD0003C / BCA1_Ch4_Serial_Amps
ALDS15MD0000C / BCA1_Ch1_Serial_Amps
ALDS15MD0004J / Airlock_CCAA_Cmd_Ack
CHAR_00000001 / Character 001 of String Parameter
ALDS15MD0002C / BCA1_Ch3_Serial_Amps
Tuesday 14
N1TP65SR0001T / Nod1_Htr3B_Temp1
ALDS15MD0003V / BCA1_Ch4_Serial_Volts
N1TP65SR0801T / Nod1_Htr7B_Temp2
N1TP65SR0401T / Nod1_Htr5B_Temp2
Airlock_CCAA_Cmd_Ack / ALDS15MD0004J
Airlock_CCAA_Cmd_Ack / ALDS15MD00... -4.C
BCA1_Ch4_Serial_Amps / ALDS15MD0003C
BCA1_Ch4_Serial_Amps / ALDS15MD0... -4.32
Airlock_CCAA_Cmd_Ack / ALDS15MD0004J
Airlock_CCAA_Cmd_Ack / ALDS15MD00... -4.C
Airlock_CCAA_Cmd_Ack / ALDS15MD0004J
ALDS15MD0004J
Inspector: Tuesday 14
Alpha Plot Info
CNT N1TP65SR0001T
CNT ALDS15MD0003V
CNT N1TP65SR0801T
CNT N1TP65SR0401T
DDD/HH:MM:SS
connected

Figure 1: An example MCT display.

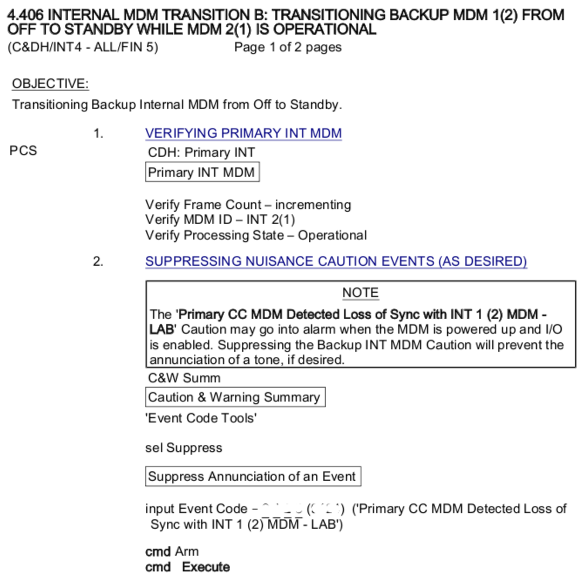
User objects User objects are things that users can manipulate and are designed to reflect their real-world counterparts. For example, a sensor on ISS might be represented as an MCT user object that contains its current value and can be displayed in many different ways. User objects are made up of MCT components, which are the programming implementation of the objects.

Manifestations User objects can be displayed in an MCT user interface in many different ways. These displays of the objects are manifestations or views. Each view refers to exactly the same underlying object.

Groups MCT displays are compositions of user objects. That is, different views of different objects can be placed together in the MCT display at the discretion of the user. Policies can restrict the kinds of compositions that are allowed to reflect organizational decisions.

Distributed object sharing User objects can be shared amongst all operators. A certain set of pre-defined user objects are always shared. Objects created by operators themselves may be shared if they want. Any changes to an object are reflected across all users and all views of the object.

Figure 1 shows an example MCT display. The left pane (Directory) is a listing of all user objects (or components) available to the operator. The middle pane (Canvas) shows groups of composed objects and their views as arranged by the operator. The right pane (Inspector) shows details of selected MCT components.



4.406 INTERNAL MDM TRANSITION B: TRANSITIONING BACKUP MDM 1(2) FROM OFF TO STANDBY WHILE MDM 2(1) IS OPERATIONAL
(C&DH/INT4 - ALL/FIN 5) Page 1 of 2 pages

OBJECTIVE:
Transitioning Backup Internal MDM from Off to Standby.

1. VERIFYING PRIMARY INT MDM

PCS CDH: Primary INT
Primary INT MDM

Verify Frame Count – incrementing
Verify MDM ID – INT 2(1)
Verify Processing State – Operational

2. SUPPRESSING NUISANCE CAUTION EVENTS (AS DESIRED)

NOTE
The '**Primary CC MDM Detected Loss of Sync with INT 1 (2) MDM - LAB**' Caution may go into alarm when the MDM is powered up and I/O is enabled. Suppressing the Backup INT MDM Caution will prevent the annunciation of a tone, if desired.

C&W Summ
Caution & Warning Summary
'Event Code Tools'

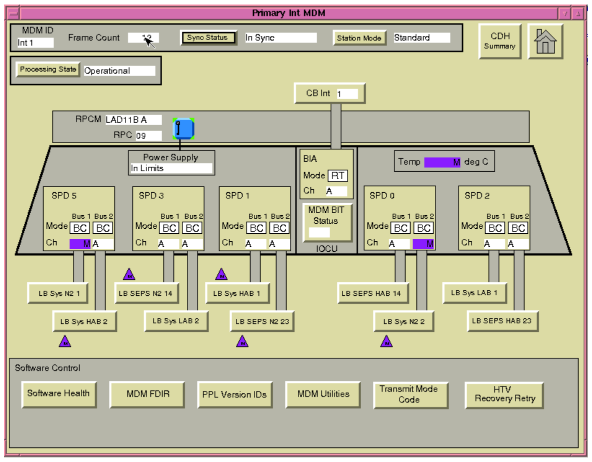
sel Suppress

Suppress Annunciation of an Event

input Event Code - [illegible] ([illegible]) ('Primary CC MDM Detected Loss of Sync with INT 1 (2) MDM - LAB')

cmd Arm
cmd **Execute**

Figure 2: An ISS procedure for transitioning MDMs showing the current display format.



Primary Int MDM
MDM ID Int 1 | Frame Count | Sync Status In Sync | Station Mode Standard
Processing State Operational
CB Int 1
RPCM LAD11B A
RPC 09
Power Supply In Limits
BIA Mode RT Ch A
Temp deg C

Figure 3: An ISS display showing MDM telemetry. This display is used while performing the MDM transitioning procedure. The upper left shows the MDM ID, Frame Count and Processing State, all of which are referenced in the procedure.

2.3 Running example

Throughout this paper, we will use an actual ISS procedure as a running example. A small snippet of this procedure is shown in Figure 2. This Figure shows the procedure as it currently appears to both ground controllers and crew. This procedure transitions an MDM (an ISS computer) from OFF to STANDBY. The first step verifies that the MDM is operational by having operator verify several different telemetry values. The second step issues some commands to turn off caution and warning messages that would result from this process. The rest of the procedure is similar, with a combination of verifying telemetry and issuing commands. This procedure document points the operator to where on their telemetry displays they need to look for a specific telemetry value and what the value should be. In the same manner, the procedure document points the operator to the specific command they need to issue. Both the telemetry values and the specific commands are on displays that are separate from the procedure document. Figure 3 shows an example of a display for ISS that would be used during performance of this procedure. Thus, the operator first looks at the procedure, then looks at a telemetry display such as this one to verify that the actual MDM telemetry matches that of the procedure.

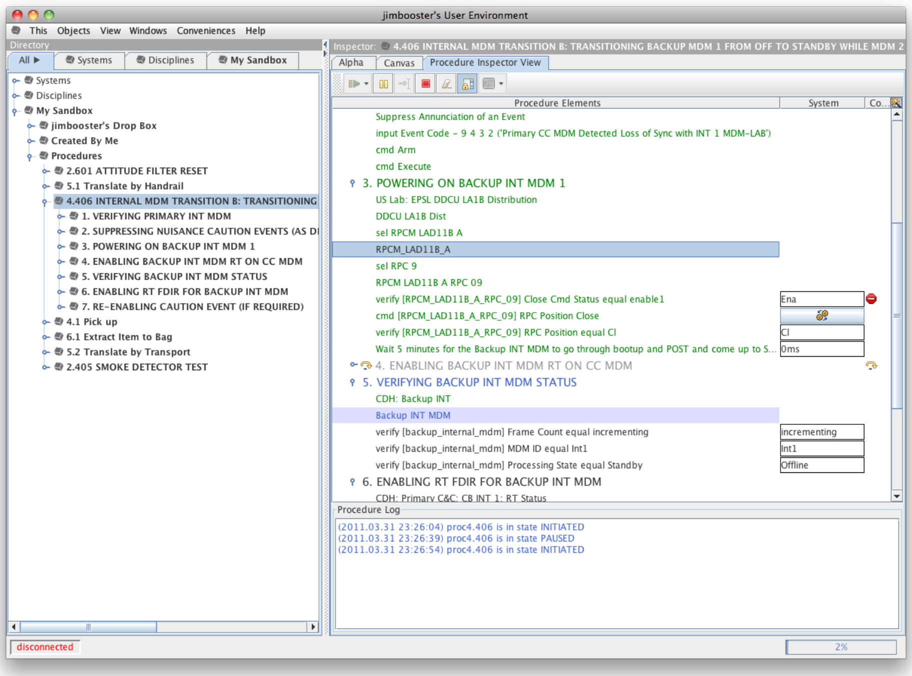
We have authored this procedure in PRL. The PRL captures not only the structure of the procedure, but also captures in a machine-understandable format, the telemetry identifiers, the comparison operator and the target value. For example, the "Verify MDM ID - INT 2" instruction in the procedure would have, in PRL, the unique identifier for the MDM ID telemetry value, a representation of the equal operator and the target value of INT 2, which is an enumeration. Thus, a computer program could check that indeed the value of the MDM ID telemetry is equal to INT 2 by comparing the current telemetry value from the ISS with the expected value in the procedure. This same is true for commands, which are represented in the PRL as well.

3 Approach

Our approach to assisting operators in procedure execution is to leverage the authoring of procedures in PRL with the new telemetry and commanding displays in MCT to provide a procedure viewer and assistant in MCT that seamlessly integrates telemetry, commanding and procedures. This approach consists of a procedure viewer that renders the PRL in a human-readable fashion within the MCT windowing environment and a procedure execution assistant that understands the machine-readable parts of the PRL procedure and can provide automation and other assistance. The key is to allow the human operator full control over the execution semantics of the procedure. This is important because the environment in which the procedure is performed is quite variable and specific parts of the procedure may need to be skipped, re-done, verified or altered based on the current execution environment.

3.1 Procedure viewer in MCT

The procedure viewer needs to render the PRL into a human-readable representation, ideally as close as possible to the traditional procedure format shown in Figure 2. Our procedure viewer implementation also has significant additional features. For example, live telemetry is embedded into the procedure display, command buttons are also embedded, a focus bar shows the current procedure execution point, collapsable steps declutter the procedure display and annotations about current procedure state are displayed. Figure 4 shows



- Systems
- Disciplines
- My Sandbox
  - jimbooster's Drop Box
  - Created By Me
  - Procedures
    - 2.601 ATTITUDE FILTER RESET
    - 5.1 Translate by Handrail
    - 4.406 INTERNAL MDM TRANSITION B: TRANSITIONING
      - 1. VERIFYING PRIMARY INT MDM
      - 2. SUPPRESSING NUISANCE CAUTION EVENTS (AS DI
      - 3. POWERING ON BACKUP INT MDM 1
      - 4. ENABLING BACKUP INT MDM RT ON CC MDM
      - 5. VERIFYING BACKUP INT MDM STATUS
      - 6. ENABLING RT FDIR FOR BACKUP INT MDM
      - 7. RE-ENABLING CAUTION EVENT (IF REQUIRED)
    - 4.1 Pick up
    - 6.1 Extract Item to Bag
    - 5.2 Translate by Transport
    - 2.405 SMOKE DETECTOR TEST

Inspector: 4.406 INTERNAL MDM TRANSITION B: TRANSITIONING BACKUP MDM 1 FROM OFF TO STANDBY WHILE MDM 2

Procedure Elements	System
Suppress Annunciation of an Event	
input Event Code - 9 4 3 2 ('Primary CC MDM Detected Loss of Sync with INT 1 MDM-LAB')	
cmd Arm	
cmd Execute	
3. POWERING ON BACKUP INT MDM 1	
US Lab: EPSL DDCU LA1B Distribution	
DDCU LA1B Dist	
sel RPCM LAD11B A	
RPCM_LAD11B_A	
sel RPC 9	
RPCM LAD11B A RPC 09	
verify [RPCM_LAD11B_A_RPC_09] Close Cmd Status equal enable1	Ena
cmd [RPCM_LAD11B_A_RPC_09] RPC Position Close	
verify [RPCM_LAD11B_A_RPC_09] RPC Position equal Cl	Cl
Wait 5 minutes for the Backup INT MDM to go through bootup and POST and come up to S...	0ms
4. ENABLING BACKUP INT MDM RT ON CC MDM	
5. VERIFYING BACKUP INT MDM STATUS	
CDH: Backup INT	
Backup INT MDM	
verify [backup_internal_mdm] Frame Count equal incrementing	incrementing
verify [backup_internal_mdm] MDM ID equal Int1	Int1
verify [backup_internal_mdm] Processing State equal Standby	Offline
6. ENABLING RT FDIR FOR BACKUP INT MDM	
CDH: Primary C&C: CB INT 1: RT Status	

Procedure Log

```
(2011.03.31 23:26:04) proc4.406 is in state INITIATED
(2011.03.31 23:26:39) proc4.406 is in state PAUSED
(2011.03.31 23:26:54) proc4.406 is in state INITIATED
```

Figure 4: A procedure viewer in MCT that supports procedure automation.

our MDM procedure in our procedure viewer. The left pane contains a list of all available procedures. Selecting a procedure displays it in the right pane. Within that pane, the procedure steps and instructions are on the left side, the live telemetry and command buttons are on the right side and the far right is used for annotations that support adjustable autonomy. Along the top is a control panel for starting, pausing and stopping the procedure. The procedure viewer is not especially relevant to a paper about automating procedures, except that the viewer needs to support a variety of adjustably autonomous interactions required to allow the human operator control over procedure execution. These are all covered in the next section, which looks at a procedure execution assistant.

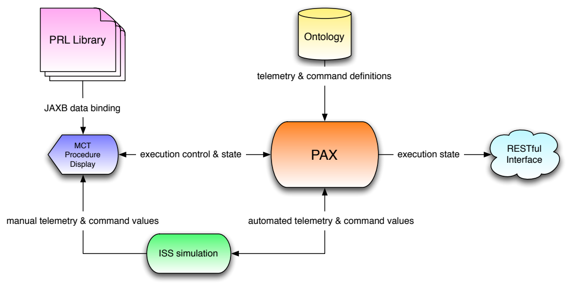

PRL Library
Ontology
telemetry & command definitions
JAXB data binding
MCT Procedure Display
execution control & state
PAX
execution state
RESTful Interface
manual telemetry & command values
automated telemetry & command values
ISS simulation

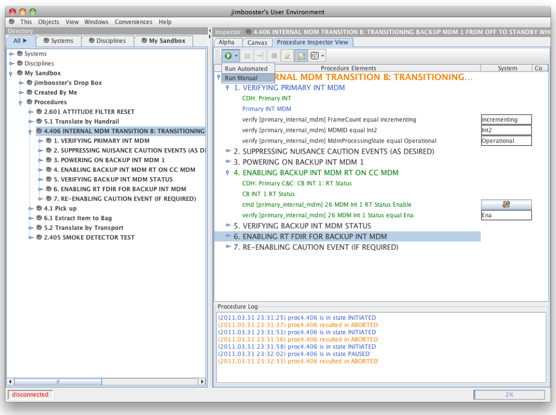
Figure 5: The overall software architecture

3.2 PAX

The Procedure Assistant for eXecution (PAX) is a software process that reads PRL and allows procedures to be both manually and automatically executed depending upon the operational constraints and the current situation. PAX interacts with the procedure display, implemented using the MCT framework, and with commanding and telemetry systems to automatically dispatch commands and evaluate telemetry under tight supervision of the operator. PAX supports both a manual and an automated mode, either of which can be chosen from the control panel at the top of the procedure.

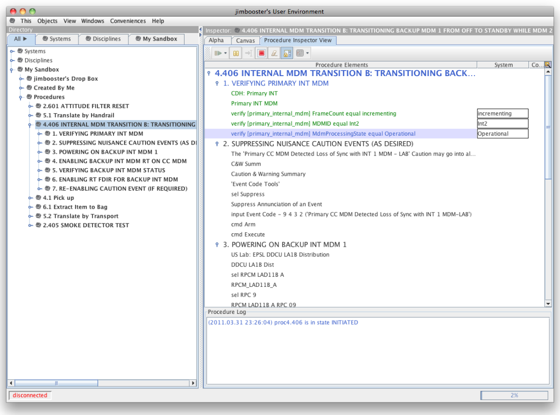
How PAX fits into the overall software architecture is shown in Figure 5. Procedure are loaded into MCT from a PRL Library. PAX finds specific command and telemetry definitions by querying an ontology. The user starts execution of a procedure by telling PAX which procedure to execute, and how to execute it. Status of PAX's procedure execution are fed both to a RESTful interface and back to MCT. PAX sends commands to and receives telemetry from the underlying system, in this case an ISS simulation.

Manual Mode In manual mode (see Figure 6), all instructions are performed and marked as completed by the operator. This is almost identical to the current operations, except that live telemetry and commands are embedded in the procedure display. Thus, if a command instruction requires that an operator send a particular command, the procedure display provides a button to send the command. However, the command is only sent if the operator clicks on the command button. PAX does provide a "bookmark" for the op-


jimbooster's User Environment
This Objects View Windows Conveniences Help
Directory
All Systems Disciplines My Sandbox
Systems
Disciplines
My Sandbox
jimbooster's Drop Box
Created By Me
Procedures
2.601 ATTITUDE FILTER RESET
5.1 Translate by Handrail
4.406 INTERNAL MDM TRANSITION B: TRANSITIONING
1. VERIFYING PRIMARY INT MDM
2. SUPPRESSING NUISANCE CAUTION EVENTS (AS DI
3. POWERING ON BACKUP INT MDM 1
4. ENABLING BACKUP INT MDM RT ON CC MDM
5. VERIFYING BACKUP INT MDM STATUS
6. ENABLING RT FDIR FOR BACKUP INT MDM
7. RE-ENABLING CAUTION EVENT (IF REQUIRED)
4.1 Pick up
6.1 Extract Item to Bag
5.2 Translate by Transport
2.405 SMOKE DETECTOR TEST
Inspector: 4.406 INTERNAL MDM TRANSITION B: TRANSITIONING BACKUP MDM 1 FROM OFF TO STANDBY WHI
Alpha Canvas Procedure Inspector View
Run Automated
Run Manual
Procedure Elements System Co...
disconnected
2%

```
RNAL MDM TRANSITION B: TRANSITIONING...
  1. VERIFYING PRIMARY INT MDM
     CDH: Primary INT
     Primary INT MDM
     verify [primary_internal_mdm] FrameCount equal incrementing        incrementing
     verify [primary_internal_mdm] MDMID equal Int2                     Int2
     verify [primary_internal_mdm] MdmProcessingState equal Operational Operational
  2. SUPPRESSING NUISANCE CAUTION EVENTS (AS DESIRED)
  3. POWERING ON BACKUP INT MDM 1
  4. ENABLING BACKUP INT MDM RT ON CC MDM
     CDH: Primary C&C: CB INT 1: RT Status
     CB INT 1 RT Status
     cmd [primary_internal_mdm] 26 MDM Int 1 RT Status Enable
     verify [primary_internal_mdm] 26 MDM Int 1 Status equal Ena        Ena
  5. VERIFYING BACKUP INT MDM STATUS
  6. ENABLING RT FDIR FOR BACKUP INT MDM
  7. RE-ENABLING CAUTION EVENT (IF REQUIRED)

Procedure Log
(2011.03.31 23:31:25) proc4.406 is in state INITIATED
(2011.03.31 23:31:37) proc4.406 resulted in ABORTED
(2011.03.31 23:31:51) proc4.406 is in state INITIATED
(2011.03.31 23:31:56) proc4.406 resulted in ABORTED
(2011.03.31 23:31:58) proc4.406 is in state INITIATED
(2011.03.31 23:32:02) proc4.406 is in state PAUSED
(2011.03.31 23:32:33) proc4.406 resulted in ABORTED
```

Figure 6: Running a procedure manually


jimbooster's User Environment
This Objects View Windows Conveniences Help
Directory
All Systems Disciplines My Sandbox
Systems
Disciplines
My Sandbox
jimbooster's Drop Box
Created By Me
Procedures
2.601 ATTITUDE FILTER RESET
5.1 Translate by Handrail
4.406 INTERNAL MDM TRANSITION B: TRANSITIONING
1. VERIFYING PRIMARY INT MDM
2. SUPPRESSING NUISANCE CAUTION EVENTS (AS DI
3. POWERING ON BACKUP INT MDM 1
4. ENABLING BACKUP INT MDM RT ON CC MDM
5. VERIFYING BACKUP INT MDM STATUS
6. ENABLING RT FDIR FOR BACKUP INT MDM
7. RE-ENABLING CAUTION EVENT (IF REQUIRED)
4.1 Pick up
6.1 Extract Item to Bag
5.2 Translate by Transport
2.405 SMOKE DETECTOR TEST
Inspector: 4.406 INTERNAL MDM TRANSITION B: TRANSITIONING BACKUP MDM 1 FROM OFF TO STANDBY WHILE MDM 2
Alpha Canvas Procedure Inspector View
Procedure Elements System Co...
disconnected
2%

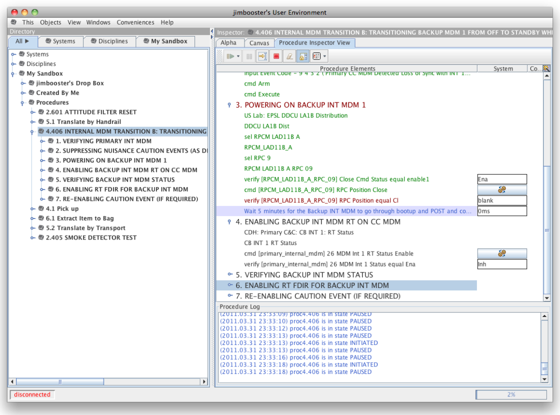
```
4.406 INTERNAL MDM TRANSITION B: TRANSITIONING BACK...
  1. VERIFYING PRIMARY INT MDM
     CDH: Primary INT
     Primary INT MDM
     verify [primary_internal_mdm] FrameCount equal incrementing        incrementing
     verify [primary_internal_mdm] MDMID equal Int2                     Int2
     verify [primary_internal_mdm] MdmProcessingState equal Operational Operational
  2. SUPPRESSING NUISANCE CAUTION EVENTS (AS DESIRED)
     The 'Primary CC MDM Detected Loss of Sync with INT 1 MDM - LAB' Caution may go into al...
     C&W Summ
     Caution & Warning Summary
     'Event Code Tools'
     sel Suppress
     Suppress Annunciation of an Event
     input Event Code - 9 4 3 2 ('Primary CC MDM Detected Loss of Sync with INT 1 MDM-LAB')
     cmd Arm
     cmd Execute
  3. POWERING ON BACKUP INT MDM 1
     US Lab: EPSL DDCU LA1B Distribution
     DDCU LA1B Dist
     sel RPCM LAD11B A
     RPCM_LAD11B_A
     sel RPC 9
     RPCM LAD11B A RPC 09

Procedure Log
(2011.03.31 23:26:04) proc4.406 is in state INITIATED
```

Figure 7: Running an automated procedure in MCT

erator to indicate their current place in the procedure. The marker may be moved to any element in the procedure by the operator. Instructions that require telemetry checks (e.g., verification instructions) will be computed by PAX and marked if the telemetry is incorrect. The operator, however, is still ultimately responsible for performing the telemetry check and marking the instruction as finished. For example, the Verify MDM ID - INT 2 instruction" mentioned in 2.3 would be visually verified by the operator. If the "MDM ID" telemetry item was not "INT 2", PAX would mark the instruction red. The operator would then set the instruction as "FAILED" by clicking the instruction and selecting "Mark as failed". If the "MDM ID" telemetry item was correct, PAX would mark the instruction green. The operator would then click the button "Mark as completed and continue to next step"

Manual mode is offered as a way for flight controllers or crew to perform procedures similar to current operations. All decisions are made by the flight controller or crew member with PAX providing some support. Procedure performance is enhanced by embedding telemetry and commands in the procedure. However, flight controllers or crew can always ignore that feature and perform the procedure using the existing command and telemetry displays as shown in Figure 3.

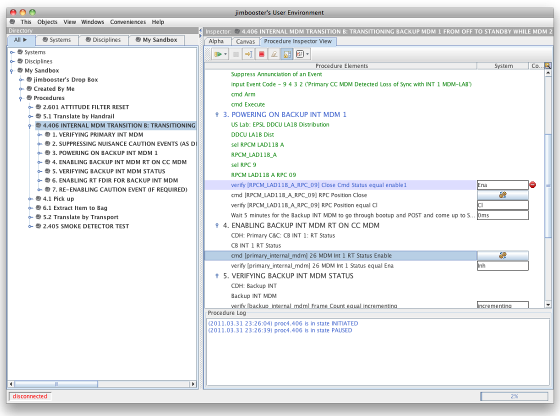
Automated Mode When a procedure is executed by PAX in automated mode (see Figure 7) , each instruction is computed and executd by PAX. Commands and telemetry are automatically issued and checked respectively. PAX detects off-nominal situations that are described in the PRL, halts execution, and marks the problematic instruction and the procedure as failed (see Figure 8) . Off-nominal situations include commands that fail to send, out of bounds telemetry values, etc. In essence, PAX is acting just as a human operator would act – checking telemetry and issuing commands. Because this is such a drastic departure from current procedure operations, we have designed an infrastructure in which the operator can have very explicit control over procedure execution by PAX. While procedures are considered the standard by which operations occur, the fluidity of space missions means that operations will sometimes need to deviate from the standard pro-


jimbooster's User Environment
This Objects View Windows Conveniences Help
Directory
All Systems Disciplines My Sandbox
Systems
Disciplines
My Sandbox
jimbooster's Drop Box
Created By Me
Procedures
2.601 ATTITUDE FILTER RESET
5.1 Translate by Handrail
4.406 INTERNAL MDM TRANSITION B: TRANSITIONING
1. VERIFYING PRIMARY INT MDM
2. SUPPRESSING NUISANCE CAUTION EVENTS (AS DI
3. POWERING ON BACKUP INT MDM 1
4. ENABLING BACKUP INT MDM RT ON CC MDM
5. VERIFYING BACKUP INT MDM STATUS
6. ENABLING RT FDIR FOR BACKUP INT MDM
7. RE-ENABLING CAUTION EVENT (IF REQUIRED)
4.1 Pick up
6.1 Extract Item to Bag
5.2 Translate by Transport
2.405 SMOKE DETECTOR TEST
Inspector: 4.406 INTERNAL MDM TRANSITION B: TRANSITIONING BACKUP MDM 1 FROM OFF TO STANDBY WHI
Alpha Canvas Procedure Inspector View
Procedure Elements System Co...
disconnected
2%

```
     input Event Code - 9 4 3 2 ('Primary CC MDM Detected Loss of Sync with INT 1...
     cmd Arm
     cmd Execute
  3. POWERING ON BACKUP INT MDM 1
     US Lab: EPSL DDCU LA1B Distribution
     DDCU LA1B Dist
     sel RPCM LAD11B A
     RPCM_LAD11B_A
     sel RPC 9
     RPCM LAD11B A RPC 09
     verify [RPCM_LAD11B_A_RPC_09] Close Cmd Status equal enable1       Ena
     cmd [RPCM_LAD11B_A_RPC_09] RPC Position Close
     verify [RPCM_LAD11B_A_RPC_09] RPC Position equal Cl                blank
     Wait 5 minutes for the Backup INT MDM to go through bootup and POST and co...  0ms
  4. ENABLING BACKUP INT MDM RT ON CC MDM
     CDH: Primary C&C: CB INT 1: RT Status
     CB INT 1 RT Status
     cmd [primary_internal_mdm] 26 MDM Int 1 RT Status Enable
     verify [primary_internal_mdm] 26 MDM Int 1 Status equal Ena        Inh
  5. VERIFYING BACKUP INT MDM STATUS
  6. ENABLING RT FDIR FOR BACKUP INT MDM
  7. RE-ENABLING CAUTION EVENT (IF REQUIRED)

Procedure Log
(2011.03.31 23:33:09) proc4.406 is in state PAUSED
(2011.03.31 23:33:10) proc4.406 is in state PAUSED
(2011.03.31 23:33:12) proc4.406 is in state PAUSED
(2011.03.31 23:33:13) proc4.406 is in state PAUSED
(2011.03.31 23:33:13) proc4.406 is in state INITIATED
(2011.03.31 23:33:13) proc4.406 is in state PAUSED
(2011.03.31 23:33:16) proc4.406 is in state PAUSED
(2011.03.31 23:33:18) proc4.406 is in state INITIATED
(2011.03.31 23:33:18) proc4.406 is in state PAUSED
```

Figure 8: A procedure failing

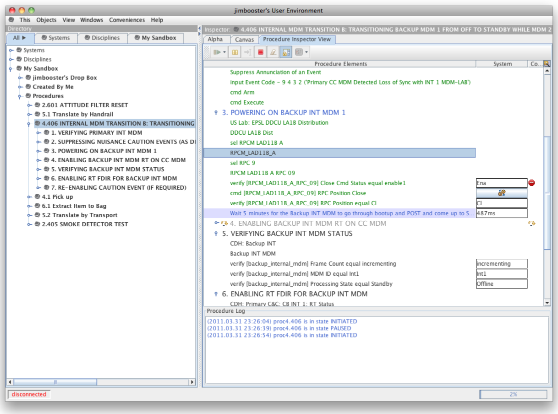
cedure. PAX is not designed to recognize these situations. In the next few paragraphs we outline some of the operator interaction capabilities built into PAX.


jimbooster's User Environment
This Objects View Windows Conveniences Help
Directory
All Systems Disciplines My Sandbox
Systems
Disciplines
My Sandbox
jimbooster's Drop Box
Created By Me
Procedures
2.601 ATTITUDE FILTER RESET
5.1 Translate by Handrail
4.406 INTERNAL MDM TRANSITION B: TRANSITIONING
1. VERIFYING PRIMARY INT MDM
2. SUPPRESSING NUISANCE CAUTION EVENTS (AS DI
3. POWERING ON BACKUP INT MDM 1
4. ENABLING BACKUP INT MDM RT ON CC MDM
5. VERIFYING BACKUP INT MDM STATUS
6. ENABLING RT FDIR FOR BACKUP INT MDM
7. RE-ENABLING CAUTION EVENT (IF REQUIRED)
4.1 Pick up
6.1 Extract Item to Bag
5.2 Translate by Transport
2.405 SMOKE DETECTOR TEST
Inspector: 4.406 INTERNAL MDM TRANSITION B: TRANSITIONING BACKUP MDM 1 FROM OFF TO STANDBY WHILE MDM 2
Alpha Canvas Procedure Inspector View
Procedure Elements System Co...
disconnected
2%

```
     Suppress Annunciation of an Event
     input Event Code - 9 4 3 2 ('Primary CC MDM Detected Loss of Sync with INT 1 MDM-LAB')
     cmd Arm
     cmd Execute
  3. POWERING ON BACKUP INT MDM 1
     US Lab: EPSL DDCU LA1B Distribution
     DDCU LA1B Dist
     sel RPCM LAD11B A
     RPCM_LAD11B_A
     sel RPC 9
     RPCM LAD11B A RPC 09
     verify [RPCM_LAD11B_A_RPC_09] Close Cmd Status equal enable1       Ena
     cmd [RPCM_LAD11B_A_RPC_09] RPC Position Close
     verify [RPCM_LAD11B_A_RPC_09] RPC Position equal Cl                Cl
     Wait 5 minutes for the Backup INT MDM to go through bootup and POST and come up to S...  0ms
  4. ENABLING BACKUP INT MDM RT ON CC MDM
     CDH: Primary C&C: CB INT 1: RT Status
     CB INT 1 RT Status
     cmd [primary_internal_mdm] 26 MDM Int 1 RT Status Enable
     verify [primary_internal_mdm] 26 MDM Int 1 Status equal Ena        Inh
  5. VERIFYING BACKUP INT MDM STATUS
     CDH: Backup INT
     Backup INT MDM
     verify [backup_internal_mdm] Frame Count equal incrementing        incrementing

Procedure Log
(2011.03.31 23:26:04) proc4.406 is in state INITIATED
(2011.03.31 23:26:39) proc4.406 is in state PAUSED
```

Figure 9: Adding a breakpoint to a procedure

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 9). 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.


jimbooster's User Environment
This Objects View Windows Conveniences Help
Directory
All ▶ Systems Disciplines My Sandbox
Inspector: 4.406 INTERNAL MDM TRANSITION B: TRANSITIONING BACKUP MDM 1 FROM OFF TO STANDBY WHILE MDM 2
Alpha Canvas Procedure Inspector View
Procedure Elements System Co...
Procedure Log
disconnected
2%

```
Systems
Disciplines
My Sandbox
  jimbooster's Drop Box
  Created By Me
  Procedures
    2.601 ATTITUDE FILTER RESET
    5.1 Translate by Handrail
    4.406 INTERNAL MDM TRANSITION B: TRANSITIONING
      1. VERIFYING PRIMARY INT MDM
      2. SUPPRESSING NUISANCE CAUTION EVENTS (AS DI
      3. POWERING ON BACKUP INT MDM 1
      4. ENABLING BACKUP INT MDM RT ON CC MDM
      5. VERIFYING BACKUP INT MDM STATUS
      6. ENABLING RT FDIR FOR BACKUP INT MDM
      7. RE-ENABLING CAUTION EVENT (IF REQUIRED)
    4.1 Pick up
    6.1 Extract Item to Bag
    5.2 Translate by Transport
    2.405 SMOKE DETECTOR TEST
```

```
    Suppress Annunciation of an Event
    input Event Code - 9 4 3 2 ('Primary CC MDM Detected Loss of Sync with INT 1 MDM-LAB')
    cmd Arm
    cmd Execute
3. POWERING ON BACKUP INT MDM 1
    US Lab: EPSL DDCU LA1B Distribution
    DDCU LA1B Dist
    sel RPCM LAD11B A
    RPCM_LAD11B_A
    sel RPC 9
    RPCM LAD11B A RPC 09
    verify [RPCM_LAD11B_A_RPC_09] Close Cmd Status equal enable1            Ena
    cmd [RPCM_LAD11B_A_RPC_09] RPC Position Close
    verify [RPCM_LAD11B_A_RPC_09] RPC Position equal Cl                     Cl
    Wait 5 minutes for the Backup INT MDM to go through bootup and POST and come up to S...   487ms
4. ENABLING BACKUP INT MDM RT ON CC MDM
5. VERIFYING BACKUP INT MDM STATUS
    CDH: Backup INT
    Backup INT MDM
    verify [backup_internal_mdm] Frame Count equal incrementing             incrementing
    verify [backup_internal_mdm] MDM ID equal Int1                          Int1
    verify [backup_internal_mdm] Processing State equal Standby             Offline
6. ENABLING RT FDIR FOR BACKUP INT MDM
    CDH: Primary C&C: CB INT 1: RT Status
```

```
(2011.03.31 23:26:04) proc4.406 is in state INITIATED
(2011.03.31 23:26:39) proc4.406 is in state PAUSED
(2011.03.31 23:26:54) proc4.406 is in state INITIATED
```

Figure 10: Skipping a step in a procedure

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 10). PAX will then not execute those in automated mode. These notations do not affect the underlying PRL, which remains intact.

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.

System Ontology A PRL file contains only part of the information necessary for autonomously executing procedures. Commands and telemetry for the underlying system are 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, 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, the operation nomenclature, the data type of the telemetry, and to what system component it belongs.

Remote access Other software processes may want to know the current state of procedure execution. Thus, PAX publishes all state information of procedure execution using JAX-RS, a Java implementation of Representational State Transfer [Fielding, 2000] or REST. The RESTful interface is essentially a web service running on the same machine as PAX. The following are valid GET 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.

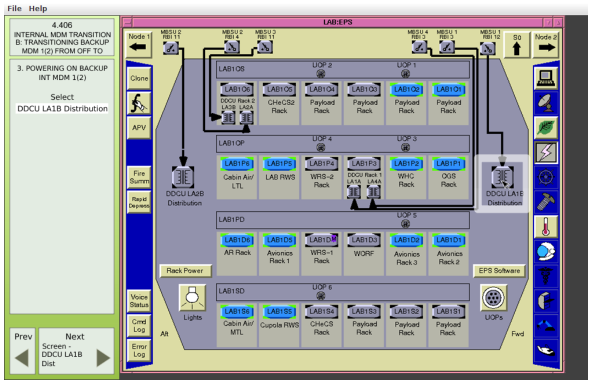
The above GET requests can also be modified with date, status, and result filters. Valid statuses for procedures are UNKNOWN, INITIATED, FINISHED, PAUSED, STOPPED. Valid results for a procedure are UNKNOWN, SUCCEEDED, FAILED, ABORTED. For example, a GET request to `/procedures?date=2010-11-04T21:55:32.346-04:00&state=FINISHED` will return all the procedures that have finished after April 11th at 21:55. The following are valid POST requests to PAX:

- `/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.

4 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 [Boddy and Bonasso, 2009] for an early version of this). We are exploring using plan recognition techniques 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 [Bonasso *et*


4.406
INTERNAL MDM TRANSITION
B: TRANSITIONING BACKUP
MDM 1(2) FROM OFF TO

3. POWERING ON BACKUP
INT MDM 1(2)

Select
DDCU LA1B Distribution

Figure 11: Auxiliary system displays integrated with procedure execution

al., 1997]. 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 [Schreckenghost *et al.*, 2008]. Lastly, we are working to integrate auxiliary system displays with automated procedure execution for further situational awareness (see Figure 11). The left pane shows the current procedure as it's being executed, while the right pane automatically brings up existing system displays as the procedure executes. The existing procedure displays are overlaid with generated annotations and highlighting (the white box), drawing the operator's eye to important information. For example, if an executing procedure affects multiple subsystems of a piece of equipment, these subsystems are highlighted in the system display for the operator.

5 Conclusion

Procedure performance consumes a significant amount of ground controllers and crew time and are a source of potential human error. We are developing a suite of tools to replace the current document-oriented procedure approach at NASA with an electronic procedure approach in which an XML file representing the procedure is interpreted for display, assistance, and execution by computer programs. The computer programs are designed to interact with the new NASA Mission Control Center (MCC) software environment. These new procedure representations and assistants seek to reduce operator and crew workload and catch performance errors.

Acknowledgments

This research is funded by NASA contract NNX10CA18C. The authors wish to thank Dr. Jeremy Frank of NASA Ames Research Center for his help in formulating (and funding) many of the key concepts of this research. NASA's former Automation for Operations (A4O) project, led by Dr. Frank, provided most of the early funding for PRL. The authors also wish to thank Alan Crocker, Brian O'Hagan, Lui Wang, and Neil Woodbury of NASA Johnson Space Center for their insights into procedure representation and execution and for their assistance with this project.

References

[Boddy and Bonasso, 2009] 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.

[Bonasso *et al.*, 1997] 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.

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

[Group, 2009] User Centered Technologies Group. Mct developers overview. Technical Report DRAFT Version 2009-04-27, NASA Ames Research Center, 2009.

[Izygon *et al.*, 2008] 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.

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

[Kortenkamp *et al.*, 2008] 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.

[Schreckenghost *et al.*, 2008] 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.

[Trimble *et al.*, 2006] Jay Trimble, Joan Walton, and Harry Sadler. Mission control technologies: A new way of designing and evolving mission systems. In *AIAA Space Operations 2006*, 2006.