ProToS: Next Generation Procedure Tool Suite for Creation, Execution and Automation of Flight Control Procedures

Thorsten Beck, Leonard Schlag, Jan Philipp Hamacher

Deutsches Zentrum für Luft- und Raumfahrt e. V., German Aerospace Center

Münchener Straße 20, 82234 Weßling, Germany

The Procedure Tool Suite (ProToS) is a software solution developed at the German Space Operation Center (GSOC) as part of the GSOC-2020 Research and Development agenda. Its purpose is to support creation and execution of Satellite Test and Flight Control Procedures and to provide an automation framework for complex operational scenarios. The software is currently in use at DLR for the AIT phase of the Eu:CROPIS project and foreseen as an essential element of the automation engine for the EDRS-C mission. This paper shall give a feature overview, present the applied design principles and the current status of development.

I. Introduction

One of the most established and commonly used monitoring and control system (MCS) for European space operations during the last decade is ESA's SCOS-2000. Although continuously modernized and adapted to new mission requirements, SCOS-2000 does by itself not support the execution of high-level flight procedures but applies a classic telecommand based approach. Conceptual extensions to this paradigm exist for some time[1] and substantial effort has gone into commercial and open-source development of procedure frameworks[2] and domain-specific languages.[3] Unfortunately, these frameworks often suffer from high complexity, high acquisition and maintenance cost or they simply do not integrate well with the products already in use by operations engineers and satellite manufacturers. Therefore, a large part of operations nowadays is still centered around the manual dispatch of command stacks and telemetry checks, requiring human decisions on the level of telemetry and command packets.

The Procedure Tool Suite developed by the Mission Control and Data System team at GSOC was designed to overcome this problem. It provides a lightweight framework for procedure development, combined with a procedure oriented commanding front-end for the core MCS. Its main purpose, however, is the support of modern procedure-based commanding concepts and the automation of complex operational scenarios in distributed environments. ProToS therefore offers a scripting engine that allows creation, management and controlled execution of mission specific automation scripts, which makes ProToS one of the most versatile tools in the GSOC multi-mission environment.

ProToS has been designed as a multi-mission framework, capable of supporting current and foreseen future missions at GSOC, starting at phases as early as AIT (Assembly, Integration and Test). It supports test execution and test documentation and has been integrated into GSOCs Central Checkout System (CCS) with the goal of providing a common procedure language for AIT and operations. It interfaces with GSOCs core MCS GECCOS,[4] an enhanced derivative of ESAs SCOS-2000 v3.1. The implementation is however generic enough for adaption to future monitoring and control systems, in particular to the upcoming European Ground Systems Common Core (EGS-CC), a software initiative led by ESA.

The ProToS software solution has been realized by a surprisingly small team, using an adapted agile development approach[5] which allowed very effective use of developer resources and short time intervals between development iterations. The framework has been developed exclusively using open-source software libraries, based on Eclipse RCP technology and licensed under the EPL.

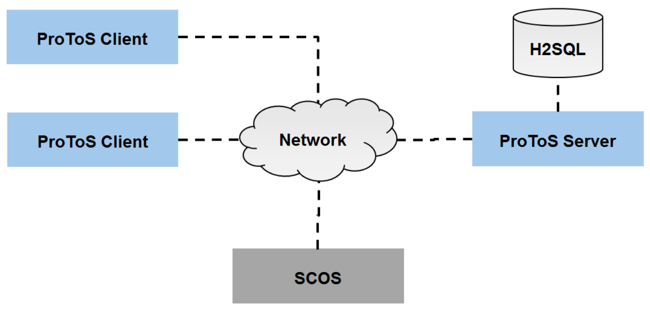
A. Range of Application

ProToS features four main components, bundled into a single software solution:

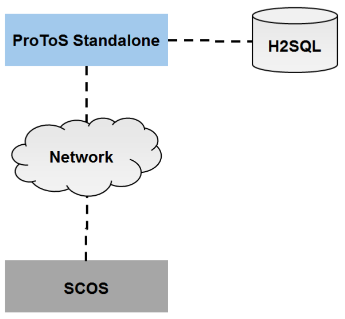
1. A central procedure and mission database repository.
2. A procedure editor offering a graphical front-end for development of Flight Control Procedures (FCPs), based on an existing TM/TC database definition.
3. A procedure executor, providing a user interface for procedure instantiation and execution, enabling the operator to interact with the procedure control flow in real-time.
4. An automation engine, which allows scheduled and event triggered execution, not only of procedures, but of arbitrary code compiled at runtime, capable of accessing procedure and system resources alike.

B. System Architecture and Deployment Scenarios

ProToS implements a three-tier architecture, featuring a central server application with a database back-end and an arbitrary number of client applications, which makes it particularly suited for work in distributed environments. When preferred, the application can also be deployed as a stand-alone desktop application for single-user environments with an integrated server.



(a) Server-Client



(b) Standalone

Figure 1. Deployment scenarios

For distributed preparation environments, the client-server architecture allows users to share a common procedure repository. Procedures can be developed collaboratively, with ProToS offering typical version control and release management capabilities. Such a setup is typically deployed in office environments where the number of clients may range from 1-20.

Releases can easily be transferred from preparation to integration or operational domains, using an XML based interface language. In a multi-user control room environment, ProToS provides a central execution and automation engine. The interface to the core MCS must be centralized and is therefore implemented in the ProToS server, which takes care of controlled procedure execution and automation. Clients in such an environment are operated in different modes: there is always only one client able to actively interact with the server, the remaining clients are operated in observer mode, granting read-only access to the repository and the procedure stack.

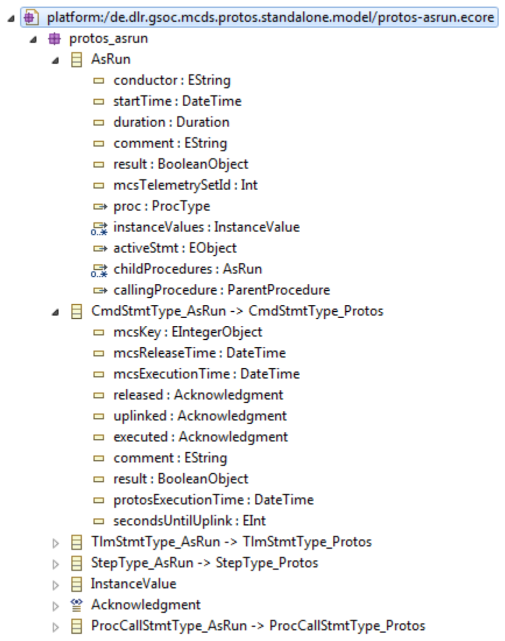
Since all server-client communication is realized via TCP/IP protocol, more complex deployment scenarios are realizable. Access to the procedure repository is not only possible in a shared office environment but also from remote sites such that a repository can be shared between partners from different organizations. As long as the network infrastructure allows it, access can be extended to the execution and automation engine, which would allow remote maintenance or even controlled access to monitoring and command capabilities from remote locations, a use case that might be desirable for payload operation scenarios.

C. Technologies

ProToS is based on the Eclipse Platform and written exclusively in Java. Eclipse, known to most as an Integrated Development Environment (IDE) for a wide range of programming languages, also provides an open source framework, commonly referred to as the rich client platform (RCP).[6] RCP enables developers to build their own applications based on the Eclipse libraries, offering a professional runtime environment and allowing highly modular designs due to the dynamic plug-in model approach. The platform allows fine-grained control of component design and behavior.

A large number of extensions and software development kits for the Eclipse Platform are freely available and ProToS makes intensive use of the Eclipse Modeling Framework[7] (EMF) for its application and runtime model. EMF offers valuable tooling for the creation of object-oriented domain models. It provides a code generator for the internal API, including adapter classes which highly simplify the creation of model-view-controler (MVC) based GUI applications and integrates seamlessly with existing Java GUI frameworks, such as SWT and JFace.

In recent years, a major revision of the Eclipse Platform has been released, known as the Eclipse 4 RCP, offering a revised framework that further improves the platform's architecture and programming model. It applies the features of EMF to allow for a more model-based application design and an improved service-oriented programming model, making use of modern design patterns like dependency injection.[8] ProToS is implemented as a pure Eclipse 4 RCP application, thereby simplifying component development and improving testability.



```
platform:/de.dlr.gsoc.mcds.protos.standalone.model/protos-asrun.ecore
  protos_asrun
    AsRun
      conductor : EString
      startTime : DateTime
      duration : Duration
      comment : EString
      result : BooleanObject
      mcsTelemetrySetId : Int
      proc : ProcType
      instanceValues : InstanceValue
      activeStmt : EObject
      childProcedures : AsRun
      callingProcedure : ParentProcedure
    CmdStmtType_AsRun -> CmdStmtType_Protos
      mcsKey : EIntegerObject
      mcsReleaseTime : DateTime
      mcsExecutionTime : DateTime
      released : Acknowledgment
      uplinked : Acknowledgment
      executed : Acknowledgment
      comment : EString
      result : BooleanObject
      protosExecutionTime : DateTime
      secondsUntilUplink : EInt
    TlmStmtType_AsRun -> TlmStmtType_Protos
    StepType_AsRun -> StepType_Protos
    InstanceValue
    Acknowledgment
    ProcCallStmtType_AsRun -> ProcCallStmtType_Protos
```

Figure 2. Extract of the EMF-based Procedure Runtime Model

In environments where it is important to work collaboratively on large shared data repositories, it is advisable to add an enterprise-suited persistence layer to EMF models. There are a number of solutions available,[9,10] most of them focusing on offline editing of shared data models. A framework that stands out in this context is the CDO (Connected Data Objects) Model Repository.[11] It offers a distributed shared model framework for EMF which provides not only a data storage back-end, but a complete server-client architecture, based on the Net4J signaling platform. CDO allows the use of pluggable data storage back-ends, i.e. relational databases. All interactions with the repository are transaction based and can be endowed with a role-based user management, allowing fine-grained access to the procedure repository. Most importantly, CDO extends the EMF change notification framework over an arbitrary set of clients, connected via TCP/IP. In fact, the ProToS three-tier architecture is made possible mainly through the technology provided by the CDO framework.

All frameworks listed here, building the basis of the ProToS application, are licensed under the EPL,[12] a commercial-friendly, open source license model. The EPL allows the commercial use of licensed frameworks and a redistribution in closed-source form, as long as the developed product builds on top of these libraries, without changing the libraries themselves. ProToS is currently distributed DLR-internally under a proprietary license.

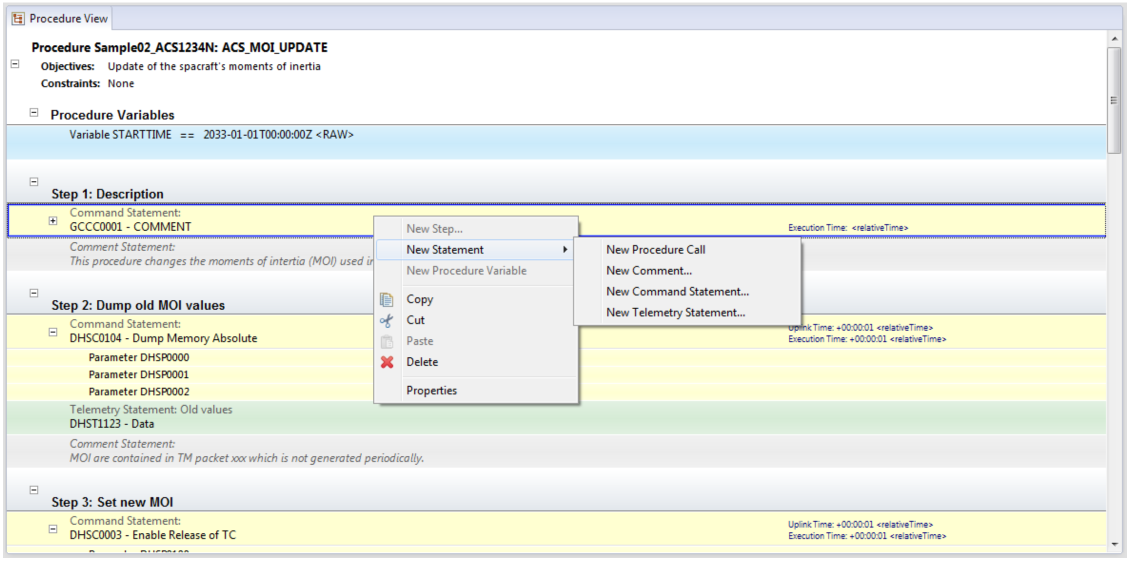
II. Application Modules

This section gives a brief overview of the three main application modules used for Procedure Development, Execution and Automation, respectively. Note that the latter component is still under active development and the description given here reflects part of the preliminary design rather than actual implementation.

A. Procedure Development

Flight Control Procedures (FCPs) in general encapsulate a set of commands, checks and decision branches which are associated with a single activity to be performed onboard a spacecraft. FCPs are typically designed far in advance and validated against simulations or engineering models.

The purpose of a procedure development tool is to support the engineer in generating an FCP implementation that is fully compliant with a given TM/TC database. ProToS therefore has the ability to import the definitions of telemetry parameters, telecommands and their associated calibrations and limits from the SCOS Mission Information Base (MIB). ProToS allows to manage more than a single MIB inside its repository and the procedure developer can switch between different database versions in a matter of seconds.



Procedure Sample02_ACS1234N: ACS_MOI_UPDATE
Objectives: Update of the spacraft's moments of inertia
Constraints: None

Procedure Variables
Variable STARTTIME == 2033-01-01T00:00:00Z <RAW>

Step 1: Description
Command Statement:
GCCC0001 - COMMENT
Execution Time: <relativeTime>
Comment Statement:
This procedure changes the moments of intertia (MOI) used i

Step 2: Dump old MOI values
Command Statement:
DHSC0104 - Dump Memory Absolute
Uplink Time: +00:00:01 <relativeTime>
Execution Time: +00:00:01 <relativeTime>
Parameter DHSP0000
Parameter DHSP0001
Parameter DHSP0002
Telemetry Statement: Old values
DHST1123 - Data
Comment Statement:
MOI are contained in TM packet xxx which is not generated periodically.

Step 3: Set new MOI
Command Statement:
DHSC0003 - Enable Release of TC
Uplink Time: +00:00:01 <relativeTime>
Execution Time: +00:00:01 <relativeTime>

Figure 3. Procedure Editor View

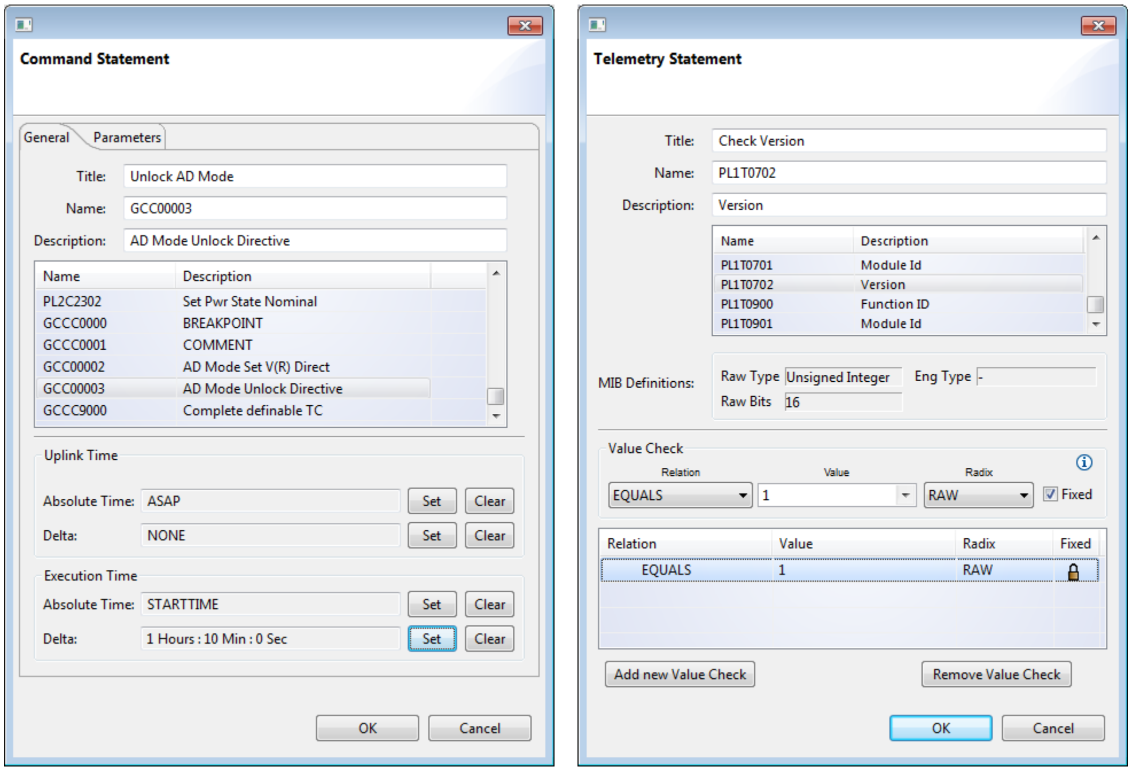
Procedure creation itself is supported by the graphical user interface providing typical editor functionality, such as adding and removing statements or common copy, cut and paste operations.

Each created procedure consists of a header, providing meta-information on author, database version and release and execution related information like duration, pre- and post-conditions. ProToS currently supports creation of the following procedure elements:

- Procedure Variables
- Steps and Substeps
- Decision Steps
- Comment Statements
- Telemetry Statements

- Command Statements
- Procedure-call Statements

Procedure variables represent placeholders that can be specified during instantiation. Procedure statements are collected in steps and sub-steps and can be further described by the use of comments. Telemetry statements may contain one or several checks on a given telemetry parameter. Command statements contain information about telecommand parameter values and their execution may be time-tagged, using either relative or absolute timestamps for uplink and execution time (Figure 4).



Command Statement

Title: Unlock AD Mode

Name: GCC00003

Description: AD Mode Unlock Directive

Name	Description
PL2C2302	Set Pwr State Nominal
GCCC0000	BREAKPOINT
GCCC0001	COMMENT
GCC00002	AD Mode Set V(R) Direct
GCC00003	AD Mode Unlock Directive
GCCC9000	Complete definable TC

Uplink Time

Absolute Time: ASAP

Delta: NONE

Execution Time

Absolute Time: STARTTIME

Delta: 1 Hours : 10 Min : 0 Sec

Telemetry Statement

Title: Check Version

Name: PL1T0702

Description: Version

Name	Description
PL1T0701	Module Id
PL1T0702	Version
PL1T0900	Function ID
PL1T0901	Module Id

MIB Definitions: Raw Type Unsigned Integer; Eng Type -; Raw Bits 16

Value Check: Relation EQUALS; Value 1; Radix RAW; ✓ Fixed

Relation	Value	Radix	Fixed
EQUALS	1	RAW	🔒

Figure 4. Command and Telemetry Statement Definition

The procedure execution flow inside FCPs can be controlled by the use of Decision Steps. When executed, these steps evaluate a boolean condition that can be related to an instance variable or to actual telemetry. Depending on the result of this evaluation, a certain path through the otherwise sequential procedure is chosen. Using Decision Steps it is also possible to pause the execution of a procedure, until the specified condition is fulfilled. Complex workflows can be facilitated by the modular approach of Procedure-Call Statements which allow nested calls to referenced procedures.

The procedure language has a clear XML representation, defined in an associated schema. This allows controlled import and export and thereby convenient exchange and distribution of FCPs, internally as well as with external partners. In order to provide backward compatibility, a number of existing export formats have been implemented, such as the MOIS XML procedure language and SCOS stack files.

The procedure editor also provides a procedure validation framework that builds upon the definitions in the TM/TC database. It supports basic live input validation and can check the prepared FCPs for semantic correctness, semantic in the sense that the all statement definitions generate a valid sequence with respect to the software's capabilities. For instance, the creation of infinite loops in the procedure workflow is prohibited. Of course, the task of validating a procedure against overall correctness is still a task that requires the work of engineers, validating flight procedures against simulations or actual hardware.

The functionality described in this paragraph is handled on client-side. The ProToS server merely provides the repository (Figure 5) and thereby the ability to develop procedures in a distributed environment. The problem of merge conflicts is circumvented by a simple locked-for-editing mechanism.

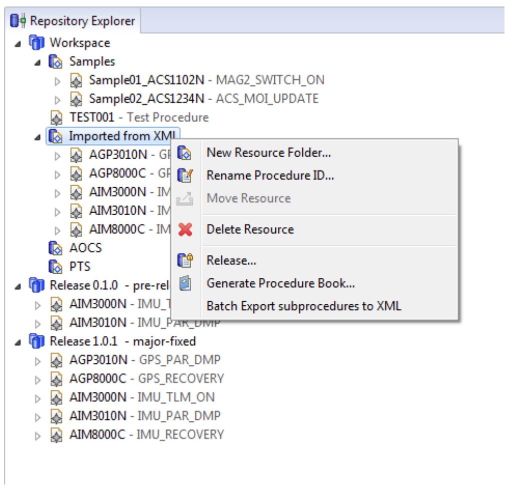


Figure 5. Procedure Repository

B. Manual Procedure Execution

The ability of ProToS to interface with GSOC's core MCS allows the controlled execution of FCPs, either in an integration or an operational environment. In the procedure execution view, a static procedure definition can be instantiated and thereby becomes a live procedure page, presenting real-time feedback for all specified telemetry statements and the uplink and execution state of command statements.

This is where ProToS derives the full advantage of the EMF/CDO notification framework. The CDO repository in this case acts as a mediator between client requests and actions and events on server side. Every change to the server runtime model is realized by a commit transaction, triggered either on server- or client-side, and every connected client is automatically notified of the update. The necessary internal function calls for this mechanism are outlined in Figure 6 for the relatively simple example of sending a connection request to the MCS.

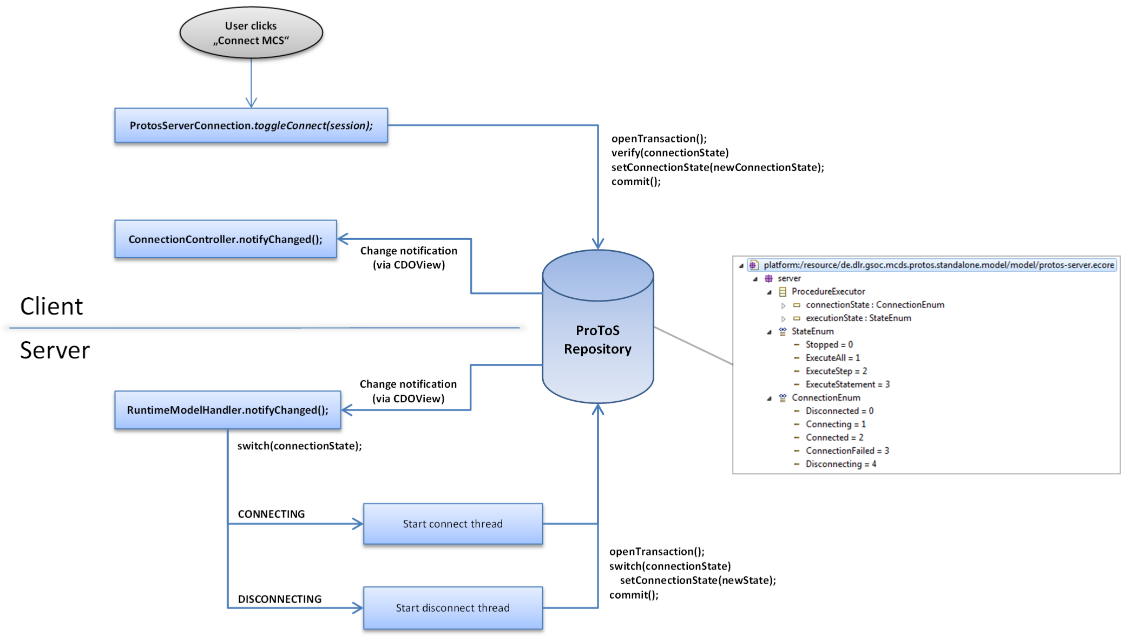
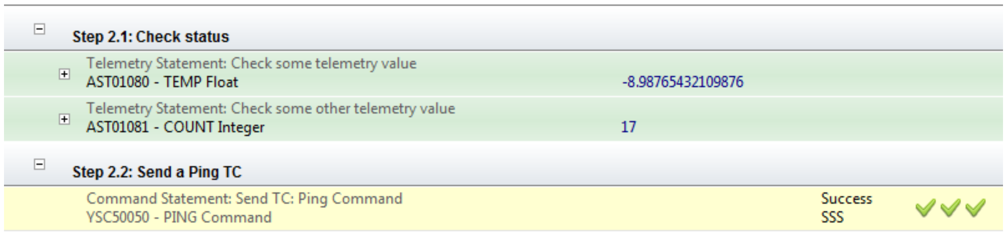

User clicks „Connect MCS"
ProtosServerConnection.toggleConnect(session);
openTransaction();
verify(connectionState)
setConnectionState(newConnectionState);
commit();
ConnectionController.notifyChanged();
Change notification (via CDOView)
Client
Server
ProToS Repository
Change notification (via CDOView)
RuntimeModelHandler.notifyChanged();
switch(connectionState);
CONNECTING
Start connect thread
DISCONNECTING
Start disconnect thread
openTransaction();
switch(connectionState)
setConnectionState(newState);
commit();
platform:/resource/de.dlr.gsoc.mcds.protos.standalone.model/model/protos-server.ecore
server
ProcedureExecutor
connectionState : ConnectionEnum
executionState : StateEnum
StateEnum
Stopped = 0
ExecuteAll = 1
ExecuteStep = 2
ExecuteStatement = 3
ConnectionEnum
Disconnected = 0
Connecting = 1
Connected = 2
ConnectionFailed = 3
Disconnecting = 4

Figure 6. Client-server interaction and notification framework, for the process of connecting ProToS with the MCS

After instantiation, the command operator can choose between different execution modes, running the procedure either in full, step-by-step or statement-by-statement. Steps are typically executed sequentially, with the currently active statement highlighted in the UI, very much like in a manual command stack, but with the advantage of supporting more complex logic based on telemetry checks and decision branches.

Through its multi-user architecture, ProToS allows simultaneous connection of an arbitrary number of clients in observer mode. This enables other users to monitor the procedure stack while it is actively



Step 2.1: Check status

Telemetry Statement: Check some telemetry value AST01080 - TEMP Float	-8.98765432109876
Telemetry Statement: Check some other telemetry value AST01081 - COUNT Integer	17

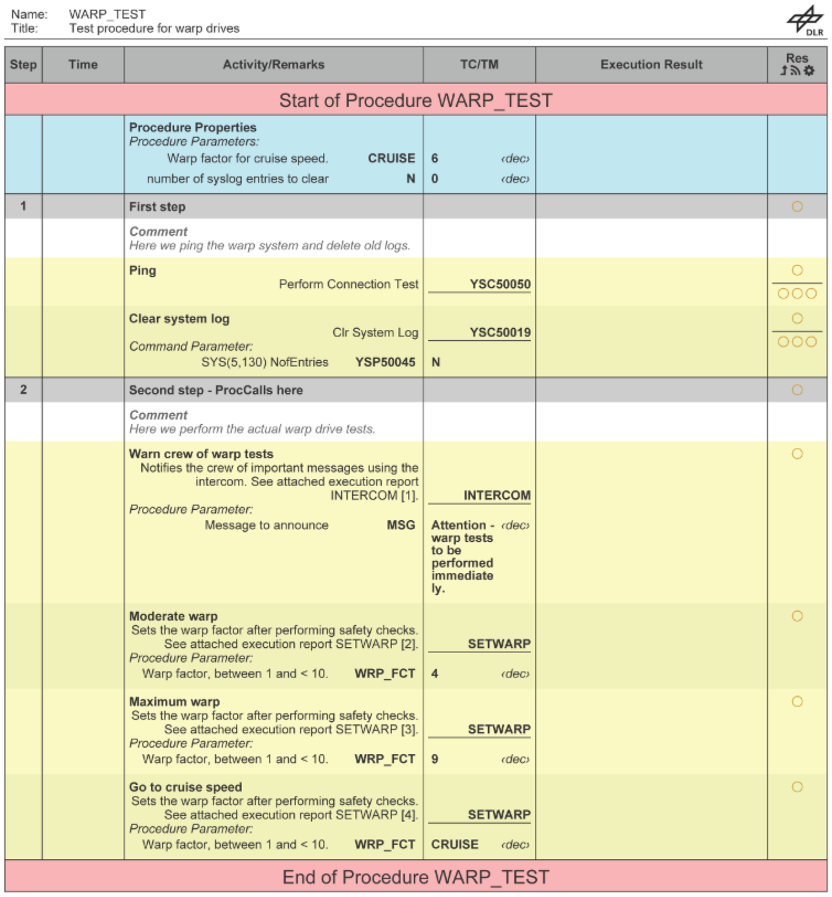
Step 2.2: Send a Ping TC

Command Statement: Send TC: Ping Command YSC50050 - PING Command	Success SSS ✓✓✓

Figure 7. Part of a Live Procedure Page with Telemetry Feedback and Execution Verification

controlled by the command operator, adding an additional level of situational awareness for the operations team.

The TM/TC interface to the MCS is realized in a dedicated ProToS server plug-in responsible for the MCS connection. It mediates actions and events between the ProToS server and the SCOS-2000 external interfaces (ExIF). The ExIF allows other applications to access core MCS components like the Event model, the Telemetry Processor or the Command Releaser. SCOS ExIF is based on CORBA, an object-oriented middleware, allowing cross-platform communication between applications, based on an Interface Definition Language (IDL). A Java implementation of CORBA and an IDL code generator is part of the Java Development Kit (JDK) since the very early version 1.3. Using CORBA, ProToS is able to inject telecommands and subscribe to updates in the telemetry or event model of SCOS-2000. Command verifications and telemetry updates are provided via call-back methods, auto-generated from the SCOS IDL definitions.



Name: WARP_TEST
Title: Test procedure for warp drives

Step	Time	Activity/Remarks	TC/TM	Execution Result	Res
		Start of Procedure WARP_TEST			
		Procedure Properties *Procedure Parameters:* Warp factor for cruise speed. **CRUISE** number of syslog entries to clear **N**	 **6** *‹dec›* **0** *‹dec›*		
1		**First step**			○
		Comment *Here we ping the warp system and delete old logs.*			
		Ping Perform Connection Test	**YSC50050**		○ ○○○
		Clear system log Clr System Log *Command Parameter:* SYS(5,130) NofEntries **YSP50045**	**YSC50019** **N**		○ ○○○
2		**Second step - ProcCalls here**			○
		Comment *Here we perform the actual warp drive tests.*			
		Warn crew of warp tests Notifies the crew of important messages using the intercom. See attached execution report INTERCOM [1]. *Procedure Parameter:* Message to announce **MSG**	**INTERCOM** **Attention - warp tests to be performed immediate ly.** *‹dec›*		○
		Moderate warp Sets the warp factor after performing safety checks. See attached execution report SETWARP [2]. *Procedure Parameter:* Warp factor, between 1 and < 10. **WRP_FCT**	**SETWARP** **4** *‹dec›*		○
		Maximum warp Sets the warp factor after performing safety checks. See attached execution report SETWARP [3]. *Procedure Parameter:* Warp factor, between 1 and < 10. **WRP_FCT**	**SETWARP** **9** *‹dec›*		○
		Go to cruise speed Sets the warp factor after performing safety checks. See attached execution report SETWARP [4]. *Procedure Parameter:* Warp factor, between 1 and < 10. **WRP_FCT**	**SETWARP** **CRUISE** *‹dec›*		○
		End of Procedure WARP_TEST			

Figure 8. A sample execution report before execution. It contains all procedure statements and a column for comments and results

It should be pointed out that the requirements to a procedure execution tool are somewhat different when comparing operational scenarios to an integration or test scenario. In a test environment for instance, the validity condition on parameter input against the TM/TC database may be relaxed or even dropped completely for the sake of failure testing. Since ProToS is applied in both domains, this possibility was considered in the design and the application behavior can be adapted via a mission-specific server configuration.

During AIT and operations alike, there is a high demand for thorough documentation of performed activities. Since all information required for such documentation is principally available in ProToS, a number of product exports have been implemented, such as the generation of procedure execution reports in formats like XML or PDF. During execution, the operator or test engineer is able to add comments to the active procedure instance, to provide additional information about the execution run (Figure 8).

C. Procedure Automation

Most operational concepts nowadays involve some level of automation. This is often motivated by the repetitive nature of manual activities but also due to high demands on the parallel execution of activities and the large amount of relevant input data. For most routine operational scenarios, the instructions given inside an FCP can be executed autonomously. A fixed set of rules thereby determines the overall execution state of an executed procedure and allows the definition of procedure entry and exit conditions as well as automated post-actions like reporting.

The ProToS automation engine supports scheduling, automated instantiation, execution and state evaluation of FCPs. The state processor has inherited its internal logic from the automation engine that is currently in use for the EDRS-A mission.[13] The final state of an automated activity is either Success, Failed or Unknown, where the latter is used in case of verification timeouts.

Since the automation of ground segment activities generally consists of more than just the autonomous execution of FCPs, more complex scenarios were foreseen and can be realized in form of automation tasks. A ProToS automation task can encapsulate basically any set of instructions that operates either on procedure or platform resources. Tasks have access to parameters provided by the MCS, to the internal procedure stack and to the file system, and they can subscribe to events or telemetry updates received from the core MCS. When triggered, tasks are able to access the procedure stack to either automatically trigger procedure instantiation and execution or to modify the execution state of currently active procedures.

Ground segment automation can be highly mission specific and it was a challenging task to align this requirement with the general concept of ProToS being designed as a multi-mission tool. The current design foresees the use of the javax.tools package, which is part of the Java Platform Standard Edition 6 (Java SE 6) and provides an API for compiling Java source at runtime. Each task then consists of a set of trigger conditions and executable instructions in form of Java source code. They are managed by the automated task manager plugin, which handles the compilation at runtime and guarantees the controlled execution, acting as a life-cycle and error handler. The execution state of automated tasks can be monitored and controlled via the client-side user interface.

The set of mission specific tasks is an integral part of the ProToS Server configuration. Each automated task has a simple XML representation which makes it very easy to load a configuration and to manage these tasks separately from the main application's code base.

III. Conclusion

The GSOC Procedure Tool Suite ProToS combines a set of flight procedure related software modules in a single framework. Its functionality covers the development, execution and automation of Flight Control Procedures, features that in the past were either non-existent or realized in heterogeneous environments, involving various tools with different programming and interface languages.

The support of procedure based commanding concepts together with an automation engine allows a transition from manual telecommand-based operations to a more high-level and largely autonomous commanding concept, as was already demonstrated for the EDRS-A mission. The inclusion of this functionality in a multi-mission tool suite now enables GSOC to apply the concept to any future mission, with EDRS-C being the next candidate.

The in-house development of a customized procedure tool allowed seamless integration with existing software and operational products used at GSOC. Its integration in the GSOC central checkout system greatly reduced effort and cost in the adaptation of FCP deliveries, through the use of well-established interface languages, while manufactures and integrators benefit from the automated execution- and test report generation.

Its multi-user architecture makes ProToS particularly suited for use in distributed environments which includes preparation, integration and control room environments. Even novel operational concepts are made possible by the ability to connect to server resources from remote sites.

The implementation was made possible by the use of powerful open-source frameworks, which allowed an efficient development process and low expected costs for software adaptations and maintenance.

References

[1]ECSS Secretariat, *Space engineering - Test and operations procedure language*, Space Engineering, No. ECSS-E-ST-70-32C, ESA-ESTEC, 2008

[2]RHEA Group, *White Paper: MOIS - Manufacturing and Operations Information System*, 2014, http://www.rheagroup.com/wp-content/uploads/2015/07/MOIS_White_paper.pdf

[3]SES Engineering: *SPELL - Satellite Procedure Execution Language & Library*, Version 2.4.4, 2015, https://sourceforge.net/projects/spell-sat/

[4]Stangl, C., Braun, A., Geyer M.P., *GECCOS - the new Monitoring and Control System at DLR-GSOC for Space Operations, based on SCOS-2000*, SpaceOps 2014 Conference, SpaceOps Conferences, (AIAA 2014-1602)

[5]Cockburn, A., *Agile Software Development: The Cooperative Game*, 2nd edition, Addison-Wesley Professional, 2006

[6]des Rivieres, J., *Eclipse Platform Technical Overview*, http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html

[7]Eclipse Consortium: *Eclipse Modeling Framework (EMF)*, Version 2.8.3 (2013), http://www.eclipse.org/emf

[8]Arthorne, J., *White Paper: e4 Technical Overview*, https://www.eclipse.org/e4/resources/e4-whitepaper.php

[9]Eclipse Consortium: *Teneo*, Version 2.0.0 (2015), http://www.eclipse.org/teneo

[10]Eclipse Consortium: *EMFStore*, Version 1.7.0 (2016), http://www.eclipse.org/emfstore

[11]*CDO Model Repository*, Version 4.4.0 (2015), http://projects.eclipse.org/projects/modeling.emf.cdo

[12]*Eclipse Public License - v 1.0*, https://eclipse.org/org/documents/epl-v10.php

[13]Beck, T., Schmidhuber, M., Scharringhausen, J.C., *Automation of Complex Operational Scenarios - Providing 24/7 Inter-Satellite Links with EDRS*, SpaceOps 2016 Conference, Daejeon, Korea (submitted for publication)