An Integrated Development and Validation Environment for Operations Automation

S.Pearson[1], F Trifin[2], S.Reid[3], W.Heinen[4]
Rhea System S.A. Avenue Pasteur 23, 1300 Wavre, Belgium

This paper will describe an ESA/ESOC led GSTP activity for an automation project (IDEA) consistent with a single integrated environment for the preparation of automation procedures. The intention is to bridge the gap between procedure definition in the offline environment and procedure validation in the execution environment. The project will capitalise on an existing generic product (MOIS) used by all missions at ESOC to create and manage procedures. It will execute these procedures automatically using the MATIS component which in turn interfaces with the ground control systems through a common Service Management Framework (SMF).

- **MATIS was initially developed on a draft of the PLUTO language, modified to support direct use of the SMF services.**
- **The SMF offers the services of all the ground control system elements via a common interface.**
- **RHEA System's Manufacturing and Operations Information System (MOIS) is used for the preparation of both manual and automation procedures.**

The main user-driven objective is to include automated operations seamlessly within the established infrastructure in order to provide a more coherent environment. The technical objectives of the project are to upgrade MATIS to support PLUTO version 1.0 of the ECSS-E-ST-70-32C (E32) standard, to develop a true PLUTO-aware editor conforming to this standard (in Eclipse RCP) and to deploy a Space System Model (SSM) consistent with the ECSS-E-ST-70-31C (E31) standard. This will facilitate an abstracted hierarchical classification of the mission data and hide the implementation details of the SMF service layer from the PLUTO scripting interface.

While this IDEA project is focused on automation procedures, a complementary MOIS and MORE project [ref[4]] will further develop a single integrated environment for all types, including manual and on-board control procedures.

The paper will present the project concepts, the design of the software and a demonstration of the new integrated PLUTO procedure preparation and automation system.

I. Introduction

THE purpose of the Mission Automation System (MATIS) is to manage and execute schedules and the PLUTO procedures they reference. These procedures can be an export product of the Manufacturing and Operations Information System (MOIS) and they can be written manually. Schedules are provided by the Mission Planning

[1] System Engineer, s.pearson@rheagroup.com
[2] Software Engineer, HSO-GIM, francois.trifin@esa.int, ESOC, Germany.
[3] Chief Technical Officer, s.reid@rheagroup.com
[4] MOIS Product Manager, w.heinen@rheagroup.com

System (MPS) – which itself receives inputs from the Flight Dynamics System (FDS) - and they can be created by the user. MATIS provides 2 applications: the Admin Manager to manage the creation and validation of the procedures and schedules, and the Operational Manager to monitor and control the executing MATIS calendar, schedules and procedures. Communication between these applications is at the file level and nominally involves the File Archive System (FARC), a simple file archive system inherited from earlier versions of the ESA Mission control System (SCOS).

MATIS communicates with the ground control systems, principally (but not limited to) SCOS and the Network Interface System (NIS) – which in turn monitors and controls the services provided by ground station equipment – via the Service Management Framework (SMF). This offers a standard interface via Application Unit (AU) adapters.

The PLUTO language (based on draft 18 of the E32 standard) was tailored to interface directly with SMF service calls. This required the introduction of SMF types in the language via the "any' data declaration and for large arrays to be populated within the scripts and passed as arguments to the SMF calls. This approach was taken so that MATIS itself need have no knowledge of the details of the SMF interface.

The MOIS operations preparation toolkit is well established throughout the European Space Industry and is the standard tool at ESA/ESOC for the preparation and maintenance of Flight Operations Plan(s), Automated Procedures and Command Sequences. It is used for the entirety of its ongoing missions. It interfaces with several mission control systems (not just SCOS) and is capable of exporting many different Operations Language scripts, PLUTO draft 18 tailored for MATIS being one of them.

From a user point of view the workflow to create, maintain and execute procedures with MATIS is too long and not transparent enough. The procedure editing and validation cycle involves iterations between MOIS, the FARC and MATIS and the resultant executable procedure is not easily recognisable from what was input by the user, making validation more difficult.

There is a need to bridge the gap between procedure definition offline and procedure validation in the execution environment. This should significantly reduce the effort, and hence the cost, of automating spacecraft operations. This project aims to implement an achievable integration of the relevant systems, such that the users are presented with a single environment providing a fully integrated tool for the definition, validation and execution of procedures fully compliant with the PLUTO standard, for automatic spacecraft operations.

II. Current state

At present the SMF interface has to be programmed directly in the PLUTO, principally because the E31 Space System Model (SSM) is not part of the solution. The PLUTO execution engine in MATIS interfaces to the SMF AU device drivers directly. These drivers connect the SMF client (e.g. MATIS) with the underlying service provider (e.g. SCOS) and define the granularity of the exposed services.

```
varSmfStringArray[1] := "1";
initiate and confirm GetLiveProvision of TmParameterProvisionServices with varSmfStringArray,
true refer by live;

varSmfName[1] := "AHT00009";
varSmfName[2] := "0";
Register of BriefTmParameter of LiveProvision of TmParameterProvisionServices with varSmfName
refer by AHT00009;

varSmfS2kTmDataFilter.InitX := true;
varSmfS2kTmDataFilter.InvalidX := true;
varSmfS2kTmDataFilter.SuperCommutationX := false;
varSmfS2kTmDataFilter.ProvisionMode := "ON_CHANGE";
varSmfS2kTmLiveOptions.ProvisionRate := 1;
initiate and confirm Start of LiveProvision of TmParameterProvisionServices with
varSmfS2kTmDataFilter, varSmfS2kTmLiveOptions refer by startLive;

initiate and confirm step refkey555

     wait until(Sampling_Time of AHT00009 > nullTime);

     if   (Result of   AHT00009).EngValue  = -69.95975954  refer by refkey555 then
```

Figure 1 Reporting Data Reference Inside Current PLUTO (SMF call)

This results in a pollution of the PLUTO scripting interface with low-level implementation detail as shown in Figure 1. This is something test and operations staff should not have to deal with. To get around this, procedures exported from the MOIS procedure development system expand the space domain elements, such as telecommands

and telemetry references, into the required SMF calls via a configurable system of templates. Procedures are therefore developed in one format and executed in a derived format. Line-by-line debugging and understanding what is happening during execution is severely hampered by this auto-generated mapping.

MATIS can execute any definition of services offered by the SMF drivers that has been provided to it in SMF service definition files. These describe the available services exposed by a single driver in terms of SMF Activity, SMF Reporting Data and SMF Events, The parameters/arguments of the available services are described in terms of SMF types, defined in XML schemas.

Clearly, from an implementation point of view, this represents a generic solution where MATIS is compatible with any upgrade of the drivers without requiring a code change; all that is needed is a replacement of the XML schema configuration files.

But from an operational point of view it means that the complexity of the drivers is shifted to the procedures. Any small change in the drivers means that the automation procedures have to be regenerated. This is of course not acceptable from a user point of view. System implementation detail such as this should not be exposed in the procedures because it makes them sensitive to changes in the software.

The SMF driver concept is suitable for complex system architecture such as the EGOS framework at ESOC, but this implementation, requiring the service calls to be embedded within automation procedures, is not at the right level. Not only is this interface too verbose, it also requires asynchronous patterns to be handled in the procedures themselves (Figure 9 below shows correct patterns for Activities). In standard E32 [ref[2]] PLUTO this is never required; the standard expects this interface to be handled by the system itself. It foresees the user programming against an abstracted set of Activities (actions that change state), Reporting Data (properties) and Events, defined within a hierarchical Space System Model [ref[1]].

The procedure development and validation process has become unnecessarily cumbersome. What is required is a multi-user integrated environment for the development and validation of automation procedures, taking advantage of modern productivity tools and plug-ins.

III. The MOIS Procedure model and its Evolution Towards PLUTO E32[ref[2]]

A key design feature of MOIS is its procedure data model.

The existing situation with MOIS and MATIS is shown in Figure 2. The MOIS model supports two user views (graphical and tabular) and provides an export in the PLUTO/SMF syntax used by MATIS.

In order to support MATIS, MOIS currently implements a complex scheme of templates and macros to embed the necessary SMF controls in the PLUTO code.

The MOIS procedure model does not support all the PLUTO constructs.

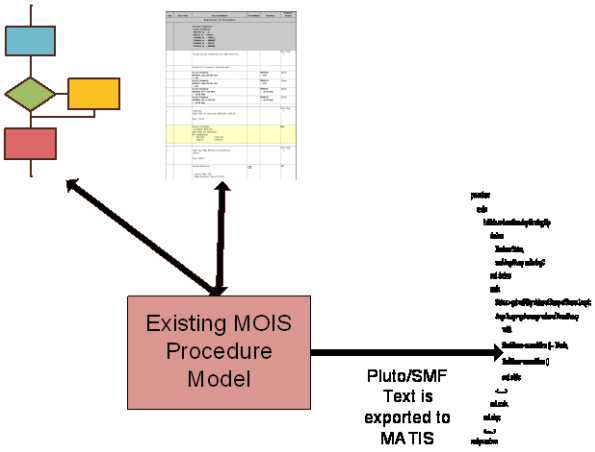


Figure 2 Current MOIS with MATIS

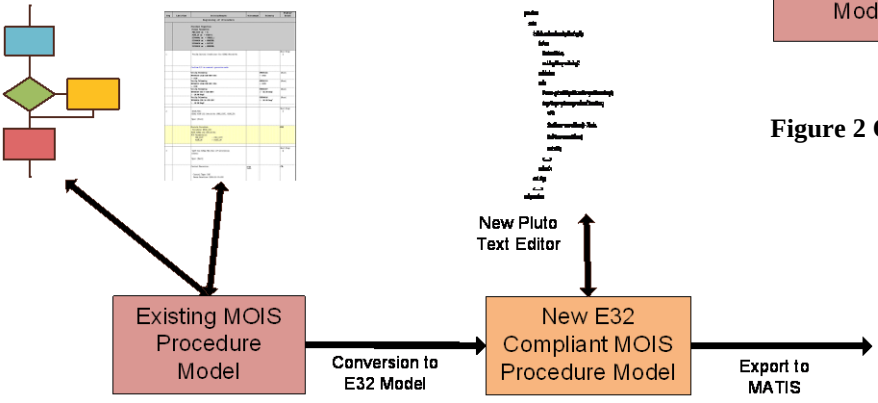


Figure 3 Intermediate Architecture

An updated architecture is shown in Figure 3. An E32 [ref[2]] compliant procedure model has been created and a new editor developed based on the established model-view design. Users can create procedures in MOIS Writer as normal and export them to the E32 format.

These automated procedures can be viewed and edited in the new PLUTO editor (although procedures created and edited there cannot be exported back to MOIS).

The approach described above - two models, side-by-side inside MOIS - is seen as a transition phase. The continued evolution taking place in the MOIS&MORE project [ref[4]] will result in the core MOIS model being E32[ref[2]] compliant with all procedure views and able to support the E32 model in full, as shown in Figure 4.

The MOIS procedure data model will therefore be updated to become E32 compliant. It will then be able to represent all PLUTO mechanisms including parallel step execution, step pre-conditions and confirmation conditions and Watchdog threads.

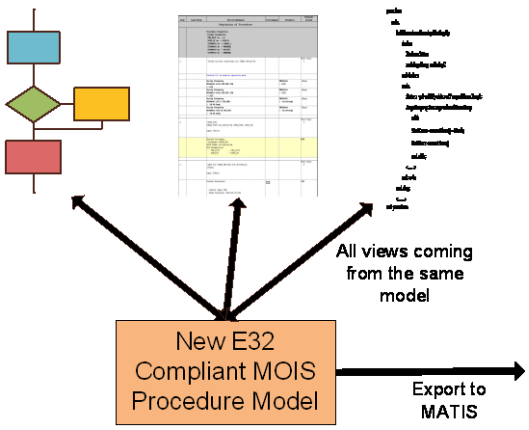


Figure 4 Final MOIS Architecture

IV. The ECSS-E-ST-70-31C Space System Model

E32 [ref[2]] PLUTO has important constructs such as the '*initiate and confirm step'* and the '*wait for event'* which obviate the need to manage asynchronous mechanisms in the script. The SMF interface, visible in the current PLUTO-MATIS, requires an interest to be registered in Reporting Data and Events which it then delivers asynchronously. These mechanisms should be managed by the execution engine and hidden from the script writer.

The SMF interface must not just be simplified, but also abstracted to the level of E31[ref[1]], exposing the 'Real-Time SSM' (as implied by the E31 and E32 standards). This interface provides services to call Activities, access Reporting Data and receive Events registered in the (static) SSM. Activities can be telecommands, procedures or a call to any other service such as a ground station control directive. Reporting Data are telemetry and other forms of (often asynchronously delivered) monitoring data. This abstraction is fundamental; Activity types are indistinguishable in the E32 PLUTO script which treats a procedure call just like a telecommand call. The execution engine is required to report back to the real-time SSM the status of the executing procedure, much like the Packet Utilisation Standard (PUS) requires a device to report back the status of an executing telecommand. The user is encouraged to think of procedures as multi-function telecommands; as just another Activity, albeit executed on the ground (in this context at least).

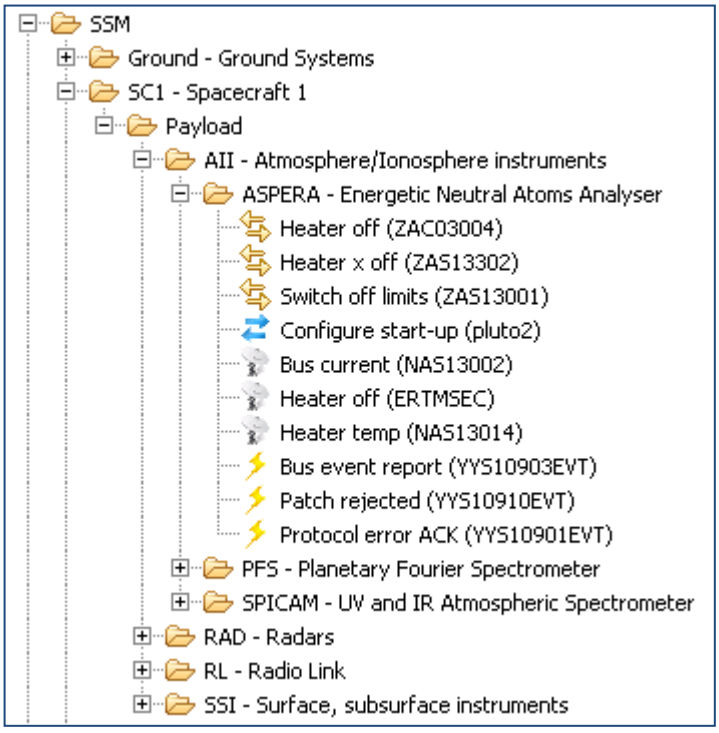


Figure 5 Space System Model

Figure 5 shows the SSM view of the space segment. At the lowest level Activities, Reporting data and Events are available for reference in the PLUTO scripts. Procedures are colour-coded to distinguish them from other Activities but they are treated identically to them.

TN2 from the CPE study [ref[3]] lists all the services required for this contract. With such an interface it is possible to write PLUTO that conforms to the true E32 definition and contains no SMF structures or mechanisms.

V. Mapping SCOS MIB and SMF Services to the SSM

The E31 Space System Model (SSM) is a delivery format which describes system configuration data in a hierarchical model. Each node is a System Element (SE) which typically corresponds to a physical entity such as a sub-system or on-board instrument. Ground systems can also be accommodated within the model, even though they are not normally strictly PUS-compliant. Data specifying the ground system services are not included in the SCOS Mission Information Base (MIB) - although this is often included in the mission databases of other control systems. The SSM therefore has a wider scope than the MIB.

Both the MIB and the SMF services can benefit from the SSM's hierarchical classification and abstractions. Mapping references to them are therefore held in the SSM, as shown in Figure 6. It should then be possible to derive an E31 delivery format from the various data sources.

Before procedures can be written for the mission this hierarchical mapping must be established as shown in Figure 7. Filters are provided to assist this process. For example, filtering by on-board APID could be a way to populate the SSM leaf nodes – if the APIDs correspond well to the on-board architecture and if this is also reflected in the SSM model.

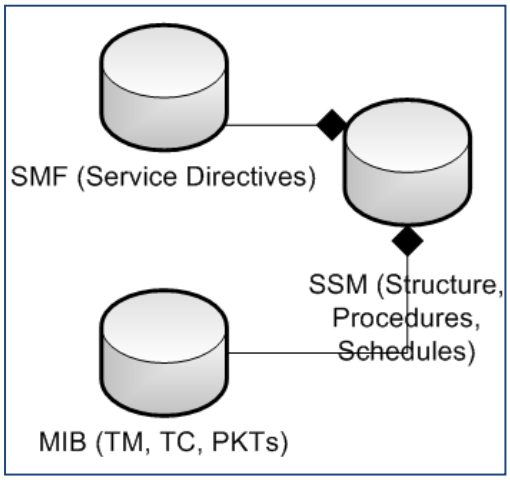


Figure 6 SSM Constituents

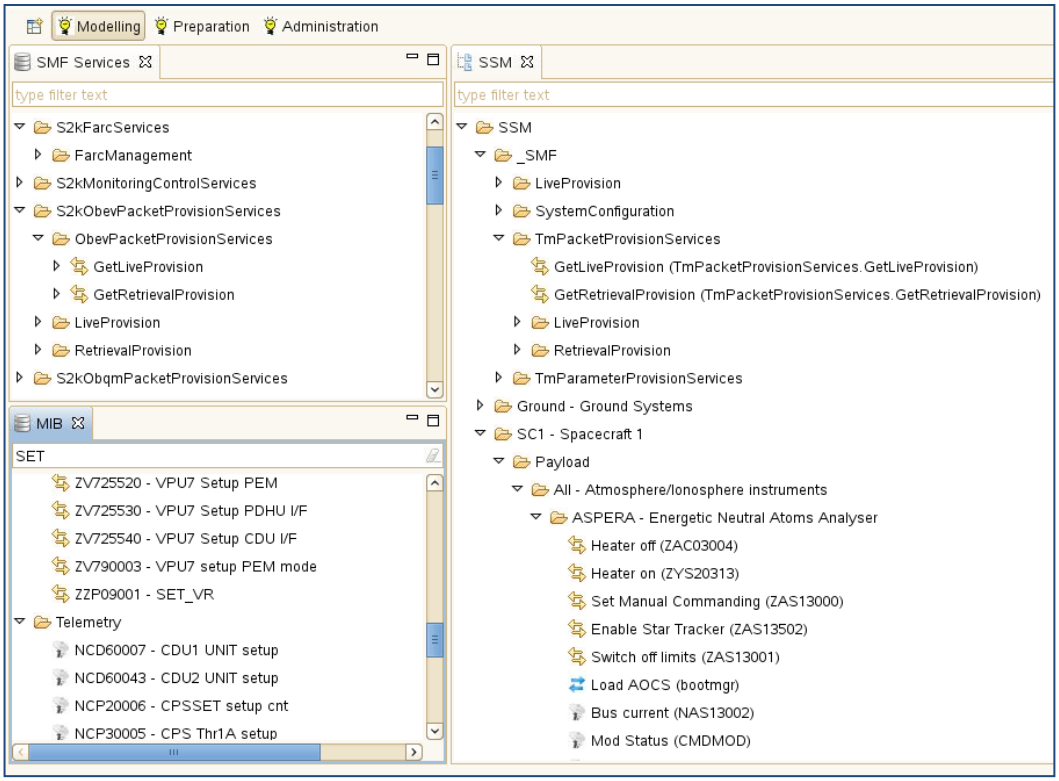


Figure 7 SMF and SCOS MIB Mapping to SSM

This model mapping and the procedures that use it must remain consistent at all times. When a new set of SMF services or a MIB is imported it must first be checked that there are no orphaned references left in the SSM. Then all procedure-Activities in the SSM must pass their compilation checks. Ensuring only controlled and consistent updates to the SSM is essential in a multi-user environment. Underneath the SSM is a version control system (VCS) which records its history and is capable of restoring earlier states.

Although a hierarchical decomposition of the space system is not absolutely necessary (many controls systems do not have one) it is fundamental to the E31 standard and cannot be dispensed with if true PLUTO procedures are to be written. The abstraction of telecommands, procedures and ground station directives to Activities means that at least some mapping data are required for the Real-Time SSM to be able to differentiate between these different types of Activity and route them to the appropriate SMF AU device driver. This breakdown is also a useful way to manage a system's complexity.

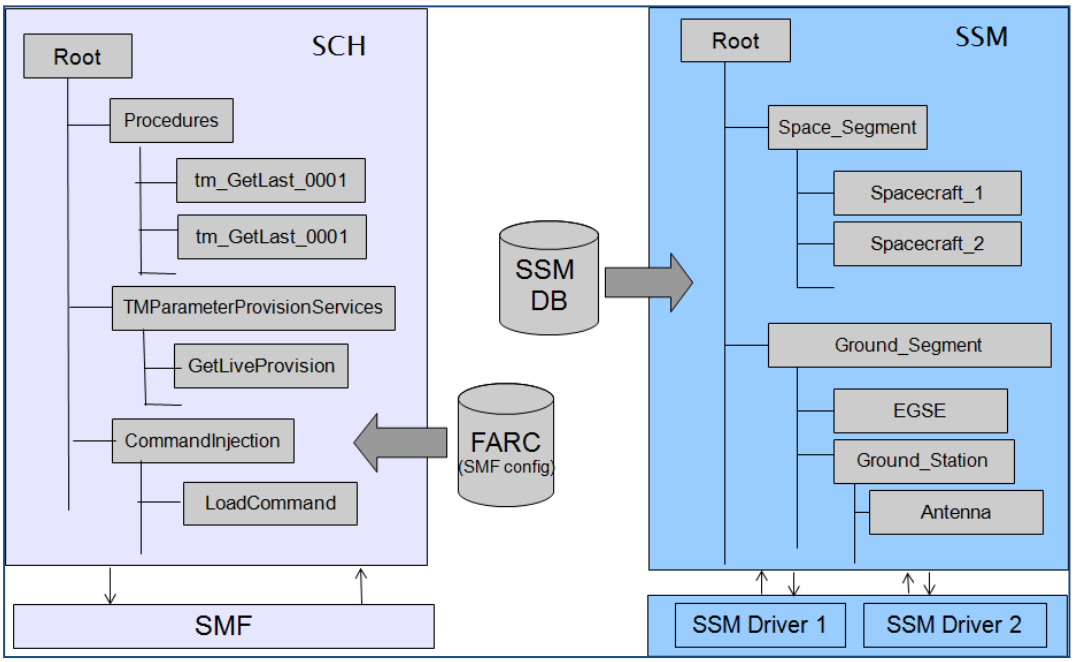


Figure 8 Current vs. New SSM Hierarchies

The SMF services are already organised in a tree structure and accessed in the PLUTO scripts via this hierarchy. However, this is primarily a functional component decomposition; a selection of services at different levels of granularity. The SSM as described in E31 is rather a hierarchical breakdown of the space and ground segments. Whereas in the SMF view all the spacecraft operations are accessed via a limited set of services provided by the control system (mostly send telecommand, get telemetry parameters and wait for packets or other Events) the SSM typically provides a breakdown of the on-board systems themselves.

Figure 8 shows how the SMF interface adapter will be modified to execute PLUTO procedures that will no longer contain SMF service calls for the space segment. Telemetry provision parameter service registrations will have to be stored as well as active telecommand references to support PLUTO coding patterns such as the two cases shown in Figure 9, both of which represent asynchronous telecommand execution. Several Activities (telecommands in this case) are sent together without waiting for each one to complete its execution before sending the next.

```
initiate TC0001 refer by act1;
initiate TC0002 refer by act2;
initiate TC0003 refer by act3;
wait until execution_status of act1 = "completed" AND
           execution_status of act2 = "completed" AND
           execution_status of act3 = "completed";

in parallel until all complete
    initiate and confirm TC0001;
    initiate and confirm TC0002;
    initiate and confirm TC0003;
    initiate and confirm TC0004;
end parallel;
```

Figure 9 Parallel Execution in the PLUTO script

In both these cases the execution component has to manage the Activity states and wait for asynchronous messages, returned via publish-and-subscribe SMF services. As can be seen, the PLUTO (user) interface remains clean and simple – and at the correct level.

VI. Using SSM References in PLUTO

The E31[ref[1]] and E32[ref[2]] standards are closely related; PLUTO scripts reference SSM items in a number of clearly defined ways. The SSM objects relevant to the procedure language are:

- Reporting Data – comprising parameters and compound parameters. A parameter is the lowest level of elementary information that has a meaning for monitoring the space system. A compound parameter is a record comprised of any sequence of parameters, arrays of parameters and sub-records.
- Activities - a space system monitoring and control function. The term Activity refers generically to procedures, telecommands (either to the space segment or to the ground segment) or any function provided by the underlying ground control systems
- Events – these are associated with System Elements, Reporting Data and Activities. An event is an occurrence of a condition or set of conditions. It is used to trigger a monitoring and control function implemented within the space system.

Each type and subtype of an SSM System Element has properties and operations that are accessible using provided services. “Initiate” and “Initiate and Confirm” are two important services for Activities which can be invoked using dedicated statements of the PLUTO language. In addition, there are two generic mechanisms to invoke other services:

- The Object Property Request - used to return a property of an object (e.g. to get the initiation time of an Activity or get the status of a printer). There is a minimum set of “standard” property requests for Activities, Reporting Data and Events.
- The Object Operation Request - used to perform an operation on an object (e.g. clear printer queue or open/close an operating system file). There is a minimum set of “standard” operation requests for Activities and Reporting Data.

Figure 10 shows how these SSM items are referenced in the PLUTO script. Friendly names assigned to the mapped items (MIB elements, SMF definitions or other procedures) make it more readable.

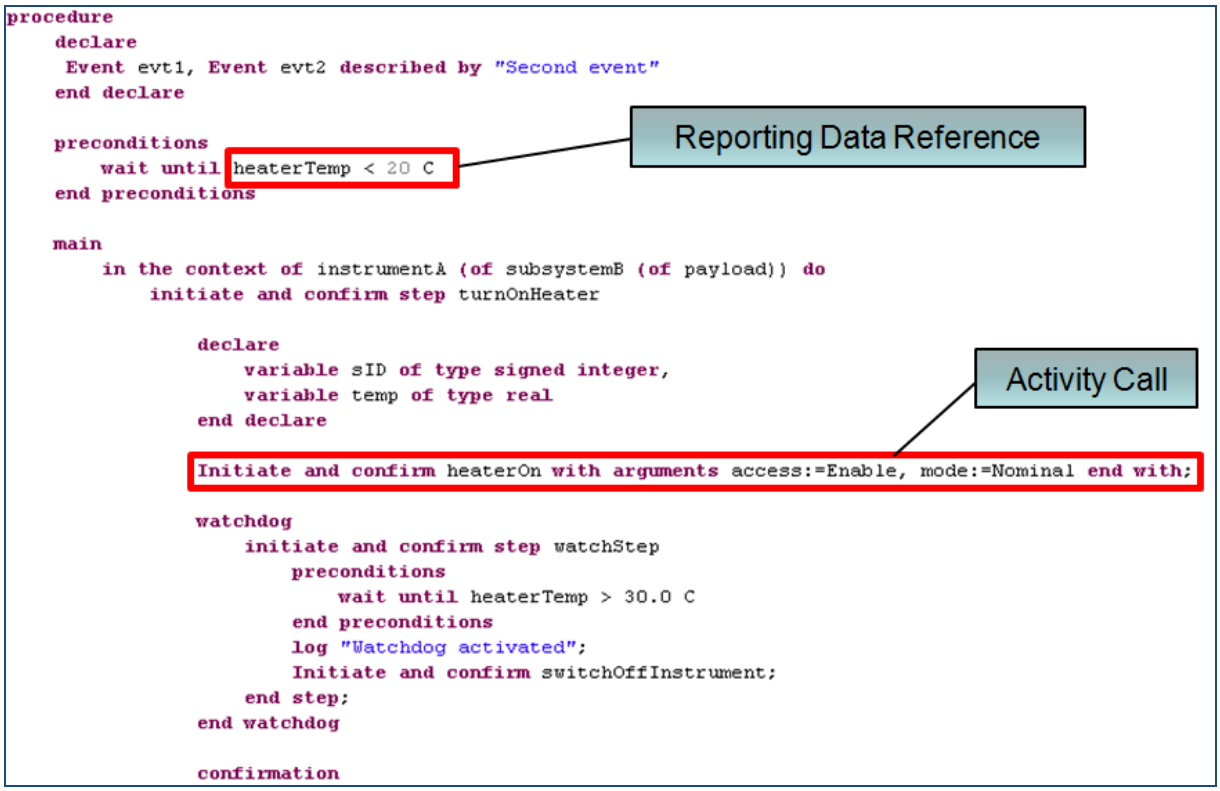


Figure 10 SSM References in PLUTO

Items can be dragged from the SSM to the PLUTO procedure. When supplementary data are required, such as telecommand parameters, a form is displayed, similar to the ones in the current MOIS editors. The items are tagged so that consistency can be maintained when something changes. During compilation these items are checked that they still exist in the SSM and that they are still consistent with their originating data source. For example, for a telecommand there are checks that the telecommand parameters exist, that they are within their range limits and that they can still be described in their specified raw format after de-calibration from their engineering values. These items can also be edited after insertion using the same data entry forms. Figure 11 shows the telecommand form that appears when an Activity with a MIB reference is dragged into the editor.

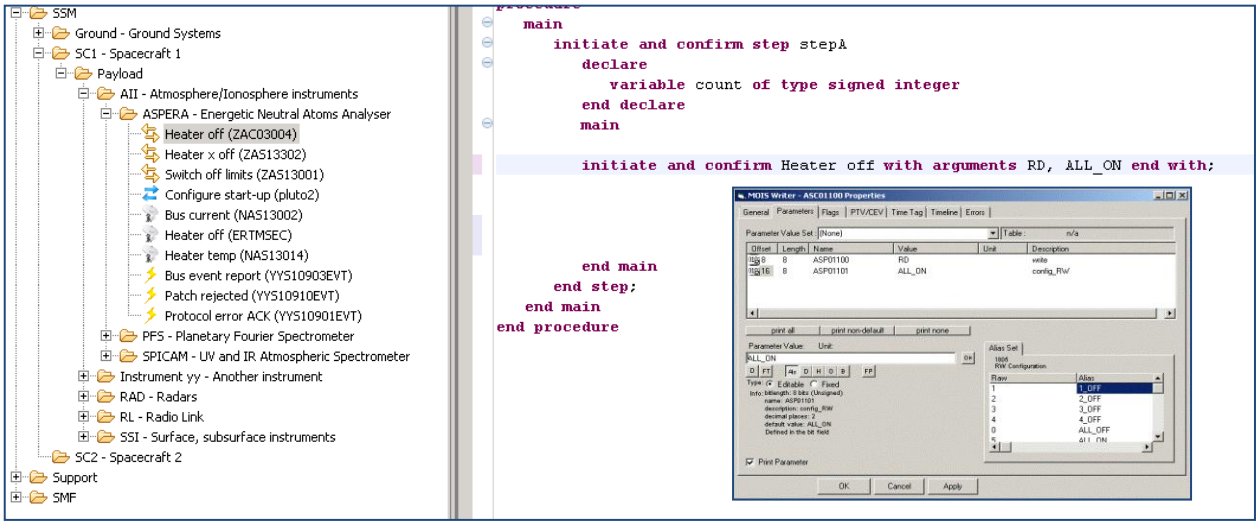


Figure 11 Drag and Drop Activity Insertion

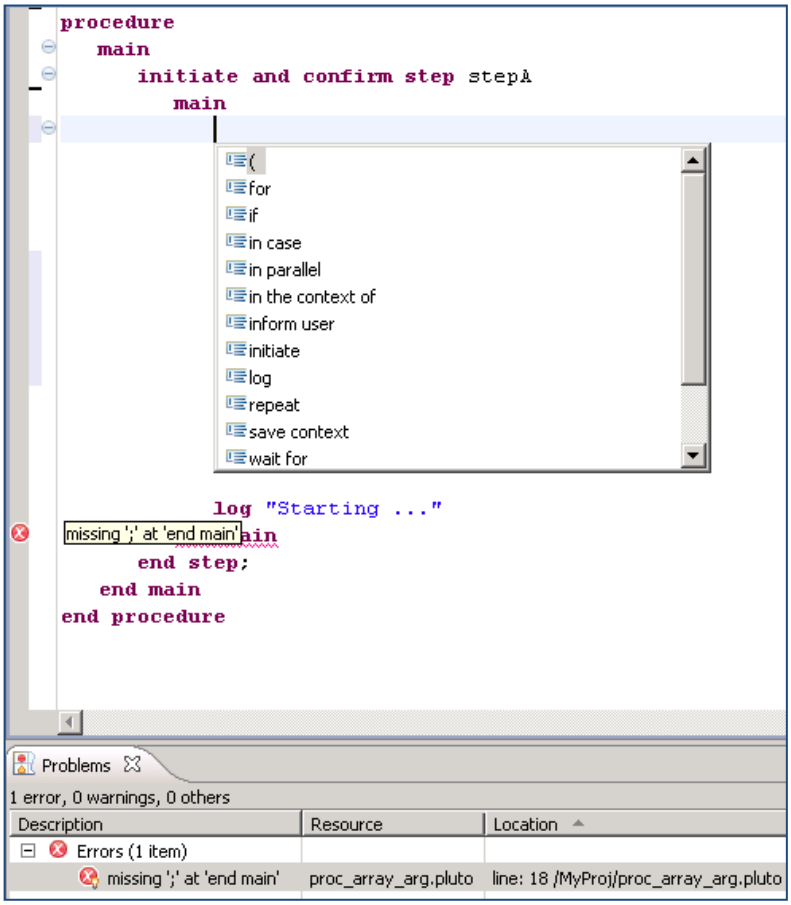
VII. A Domain Specific Language Editor

The PLUTO editor is based on the Domain Specific Language (DSL) framework Xtext, provided as part of Eclipse.

> Xtext is a framework for development of programming languages and domain specific languages. It covers all aspects of a complete language infrastructure, from parsers to linker, compiler or interpreter, to full Eclipse IDE integration. It comes with good defaults for all these features and all aspects can be tailored to specific needs.

Using a formal grammar as input, Xtext provides a full implementation of the language. Its compiler components include a parser, a type-safe abstract syntax tree (AST), a serialiser and code formatter, a scoping framework and linking, compiler checks with static analysis and a code generator or interpreter. These runtime components integrate with and are based on the Eclipse Modelling Framework (EMF). It creates an Ecore model which can be used with other EMF frameworks such as the Graphical Modelling Project GMF.

Figure 12 shows some aspects of Xtext when configured with the PLUTO grammar. The grammar is checked dynamically and annotations appear in the margin explaining any discrepancy. These errors are also listed in the Problems view. This mechanism is extendable beyond the syntax and will be used to display and locate semantic errors which are detected during compilation. For example consistency checks against referenced Activities and Reporting Data



Xtext provides:

- Syntax Colouring – based on the lexical structure and semantics
- Code completion – the editor proposed valid code completions at any point, helping with the syntactical details of PLUTO
- Validation and Quick Fixes – these can be added to tackle errors and warnings so that they can be corrected with a single keystroke
- Bracket and keyword matching
- An Outline view – showing the semantic structure
- Code formatting – to ensure proper and consistent indentation and formatting making all procedures readable.
- Template Proposals – for standard tasks
- Rename Refactoring
- Folding
- Hyperlinks for Cross References
- Comment toggling

It has the look and feel of the Eclipse Java editor but it is also very customisable.

VIII. Features of PLUTO v1.0

PLUTO has some interesting and some new features. Part of this project is to upgrade the PLUTO execution engine so that it properly supports v1.0 of the standard.

A. Engineering Unit Conversions

PLUTO supports variable definitions with engineering units. Arithmetic operations are type checked at compilation time. Only compatible unit conversions are allowed.

```
declare
   variable length of type real with units m,
   variable speed of type real with units m/s,
   variable time of type real with units ms
end declare

length := 5 m + 10 kg;  // error
speed := length / time; // ok
```

Automatic conversion between compatible units is then performed at runtime.

```
declare
   variable length of type real with units m,
   variable speed of type real with units m/s
end declare

length := 1 m;       //value of length is 1000m
length := 3 ft;      //value of length is 0.914 m
speed := 3.6 km/h;   //value of speed is 1 m/s
```

Engineering values are defined with units in the SCOS MIB, This language feature can be used to ensure type safe storage and manipulation of compatible telemetry and telecommand parameters.

B. New in PLUTO v1.0

The main differences between the draft used by the current MATIS system and version 1.0 of the E32 standard are:

- Syntax changes - most notably the variable declaration change:

```
declare
   real x units m,                          // old
   variable x of type real with units m     //new
end declare
```

- The Context statement which defines the SSM search context in which the enclosed set of statements will be executed. It is a useful shorthand so that the full SSM path does not have to be specified for every Activity or Reporting Data reference.

```
in the context of ASPERA of AII of Payload of SC1 of SSM do
    log "Bus current = ", BusCurrent;
end context;
```

- Enumerated types – a set of enumerated values can be defined:

```
declare
   enumerated Colour("red", "green", "blue")
end declare
```

- For a Property data type variable declaration a previously defined SSM object property can be inherited:

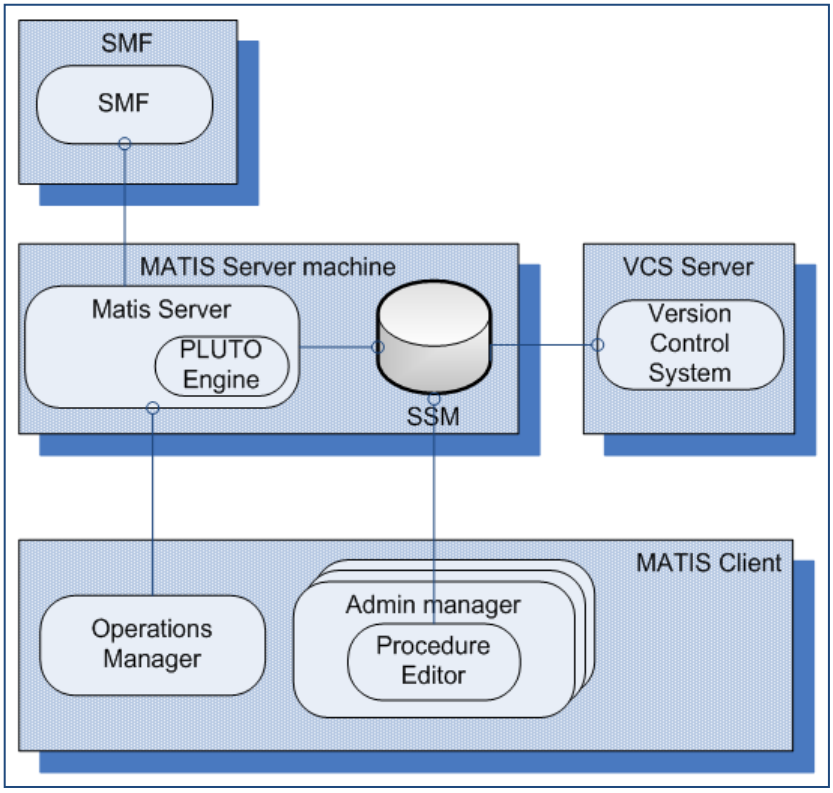
```
declare
   variable x of type same as Voltage of Battery_1
end declare
```

IX. Architecture and Technology

The existing MATIS system has been designed to execute procedures in operational schedules. It separates schedule monitoring from procedure and schedule development. Two other components – MOIS for procedure editing and the FARC for procedure storage – are involved in the procedure lifecycle making development and validation less than optimal. An aim of the project is to make this process more integrated and therefore easier to use.

The SSM is the central information provider in this multi-user environment and must therefore remain consistent at all times. Updates committed to the SSM are kept under version control so that a history is maintained and earlier baselines can be retrieved. In the operational environment SSM baselines, which have been delivered from the development environment, cannot in general be modified (this is an operational constraint and not a limitation of the software).

In this architecture, external clients such as the existing MOIS editors can also access the SSM and VCS servers. They too will be able to write automation procedures against the SSM and update the SSM with new procedure Activities. As described earlier, these editors will only be able to use a subset of the features of PLUTO until MOIS upgrades its object model to one based fully on E32.



X. Conclusion

This paper has described the ongoing ESA GSTP project called “IDEA” whose objective is to streamline and improve the usability of EGOS automation. It will do this by providing an integrated user interface using modern plug-ins. In the process it will implement the ECSS standards as they were intended by introducing an SSM and it will hide the SMF system level interface within the system where it belongs.

The project is being developed using Scrum, an iterative and incremental Agile software development method, in close collaboration with the users. This is proving to be a useful and flexible approach with new ideas constantly tested against users’ needs.

It paves the way for MOIS to be updated to write procedures against the SSM and eventually to base its procedure data model on the E32 standard in order to support all of PLUTO’s automation mechanisms.

Appendix A
Acronym List

AGILE	Class of evolutionary software development methods http://agilemanifesto.org/
APID	Application Process Identifier (see PUS)
AST	Abstract Syntax Tree
DSL	Domain Specific Language
E31	[ref[1]],
E32	[ref[2]],
Ecore	EMF meta-model
EGOS	ESA Ground Operations Software
EMF	Eclipse Modelling Framework http://www.eclipse.org/modeling/emf/
ESOC	European Space Operations Centre, Darmstadt
FARC	File Archive (part of EGOS)
FOP	Flight Operations Plan
GMF	Eclipse Graphical Modelling Framework http://www.eclipse.org/modeling/gmp/
IDEA	Integrated Development and validation Environment for Operations Automation
MATIS	Mission Automation System (part of EGOS)
MIB	Mission Information Base (see SCOS)
MOIS	Manufacturing and Operations Information System
MPS	Mission Planning System
ORM	Object Relational Mapping
PKT	Packet (see PUS)
PLUTO	Procedure Language for Users in Test and Operations (see E32)
PUS	ESA Packet Utilisation Standard ECSS-E-70-41A
RCP	Eclipse Rich Client Platform http://www.eclipse.org/rcp/
S2K	See SCOS
SCH	Service Consumer Handler – see MATIS
SCOS	ESA Mission Control System
SCRUM	Agile development method with ‘sprints’
SE	System Element – see E31
SMF	Service Management Framework – part of EGOS
SPRINT	One of a series of short lifecycles in Scrum development
SSM	Space System Model – see E31
TC	Telecommand
TM	Telemetry
VCS	Version Control System
Xtext	Eclipse DSL framework http://www.eclipse.org/Xtext/

XI. References

[1] Space engineering, Ground systems & operations – Monitoring and control data definition E31: ECSS-E-ST-70-31C, 31 July 2008

[2] Space engineering Test and Operations Procedure Language E32: ECSS-E-ST-70-32C, 31 July 2008

[3] CPE Study TN3. ECSS extensions and language conversion RHEA.CS1097.DOC.03, 2010-08-04

[4] Mission Operations Preparation Environment: A new approach for the future, W. Heinen, S. Reid, S. Pearson , SpaceOps 2012

[5] The PLUTO operations procedure language and its use for RADARSAT-2 mission operations, Mark A. Seymour, MacDonald Dettwiler and Associates, SpaceOps 2004, 17 –21 May 2004

[6] AIAA 2006-5691 - EGOS Architecture and the Impact on Existing ESOC Ground Data Systems, SpaceOps 2006, 19 - 23 June 2006

[7] AIAA 2006-5532 - Automation of ESOC Mission Operations, SpaceOps 2006, 19 - 23 June 2006

[8] AIAA 2006-5613 - Advantages of Using SSM in the Frame of a Mission Preparation Environment, Stefano Mattia, Massimo Argenti, Mauro Cardone, Dataspazio S.p.A. &Agenzia Spaziale Italiana, SpaceOps 2006, 19 - 23 June 2006

[9] AIAA 2008-3340 - Mission Automation Infrastructure Tools at ESOC, A. Loretucci, D. Di Nisio, S. Haag, M. Pecchioli, P. Bargellini, T. Nogueira, SpaceOps 2008, 12 - 16 May 2008

[10] AIAA 2008-3417 - ECSS E-70-32 Test Platform Features and Applicability Area - F. Croce and A.Simonic, Vitrociset PEG, SpaceOps 2008, 12 - 16 May 2008

[11] AIAA2010-1968 - ADM-AEOLUS: Mission Planning Re-use, Autonomy and Automation, SpaceOps 2010, 25 - 30 Apr 2010

[12] AIAA 2010-2308 - ESOC Earth Observation Missions and the Automation of Operational Routine Tasks, SpaceOps 2010, 25 - 30 Apr 2010