AN ECSS-E-70-32 COMPLIANT ENVIRONMENT WITH EVOLUTION CONSIDERATIONS

Francesco Croce[(1)]

[(1)] *EUMETSAT*
Eumetsat Allee 1 - D-64295 Darmstat - Germany
Francesco.croce@eumetsat.int

ABSTRACT

Within the EUMETSAT Monitoring & Control Applications testing infrastructure the ECSS-E-ST-70-32 specifications and related ECSS-E-ST-70-31 dependencies, have taken as reference for the test procedure language implementation and system elements modelling.

While the PLUTO syntax as defined in the E-70-32 issue 1 (Annex A) has not been adopted, the E-70-32 semantics and runtime specifications have been applied since considered flexible and generic to be used in an applications testing environment.

The basic E-70-32 semantics specifications have been enhanced with concepts driven by lessons learns from existing developed E-70-32 systems and also generally considered missing from the standard. Typical enhancements have included: complex data types management, exception handling and annotations. The activities have covered: semantics model definition and description mapped to an XML schema and subsequent implementation compliant to the defined model. The resulting implementation has been a procedure language environment having two fundamental characteristics: compliance to E-70-32 - with relevant extensions - and the availability of a generic model on top of which "language views" can be developed with each view supporting the model full set of features or a level of tailoring according the domain the language environment is required to be used or operated.

In addition to the procedure language implementation, the adoption of a static and runtime Space System Model (SSM) has been specifically addressed with the runtime SSM having the role of modelling infrastructure of all involved runtime entities through a flexible and modular architecture.

The presentation aims to present the procedure language and SSM environment with possible ECSS-E-70-32 evolution considerations.

INTRODUCTION

The corrective and evolutive software maintenance of M&C applications of EUMETSAT missions is managed by of a single organisational section (M&C Applications Team) part the Maintenance and Engineering Division within the OPS department.

In order to improve all aspects related to system verification process involved in the applications maintenance lifecycle, the M&C team is sponsoring the definition and implementation of a generic system testing and verification infrastructure with the high level goals

- To streamline and harmonise the process based on a common infrastructure and testing model
- To allow to define, execute and assess test sessions with a high level of automation

The automation capability is of paramount importance in support to a fast and yet formal assessment of a new application versions vs. the previous requirements baseline and to verify that no regressions have been introduced by the new changes.

Fundamental required features and capabilities of the infrastructure have been specified in terms of:

- Functional Integrated environment: in support to the verification process with executable test plan definition and validation, test execution and post analysis assessment of pass/fail criteria associated to SUT formal requirements (e.g. Software Requirements Specifications)
- Automation: referring to the need to automate all repetitive low-level tasks (e.g. test bed set-up), test execution, with assessment of pass/failed criteria and automatic test documentation production
- High and flexible customisation: supporting different types of System Under Test (SUT) interfaces (technology, mechanisms, data types); interface to emulator/simulators; interfaces to external tools used for test definition, management and post analysis; language and engine extensions
- Formal data-model definition and handling: covering simple data types and complex types with possibility to extend the model with custom types (user defined)
- Maximum usage of space and technology standards: allowing to reduce costs/effort in the definition of fundamental aspects relying on standardisation bodies and technologies boards as well as possibility to take advantage of standards evolution driven by the user community
- Adoption of formal components based technology: such as the infrastructure can be extended though components implementation as add-ons (plug-in) to the infrastructure
- Maximum adoption of open source technologies: with very limited or no vendors dependency
- Support to multi-platform virtualised environments

In terms of standards the following have been adopted:

- ECSS E-ST-70-31C (E-70-31) Space System Model as common semantic and runtime technology
- ECSS-E-ST-70-32C (E-70-32) with enhancements as procedure language definition and execution engine

The ECSS standards adopted concepts and introduced enhancements are the subject of this paper and reported in the following sections with particular reference to the procedure language capabilities. The paper assumes knowledge of the standards high level concepts. References are reported in [1] and [2].

E-70-31 APPLICABILITY

Within the test infrastructure the E-70-31 Space System Model is applied as common semantic model, supported at runtime with the availability of a Runtime SSM (RT-SSM).

The SSM is a means to organise and capture space system information and knowledge in terms of ***system elements*** (SE) ***activities*** **(**Act**)** ***reporting data*** **(**Rdt**)** and ***events*** **(**Evt**).**

A functional entity may be represented by a *domain-specific view* in the SMM, with each view modelling the particular **domain of interest** of the entity and **specific functional application and/or behaviour** associated to the entity.

The above concepts can be considered part of a generic semantic model that can be used and instantiated to a particular context of SSM applicability. The adoption of common semantic allows:

- To rely on an abstract layer above low-level technologies, decoupling technologies from meaning. This has not only conceptual value but the approach is used at application design
- Encourage the interoperability of entities definition according to a common semantic

Within the test infrastructure the SSM semantic is used as common paradigm to model all test artefacts and involved elements such as test plan, SUT but also infrastructure functionality like activities (telecommand, procedure, system scripts) executor with exposed executor activities such as as StartActivity, StopActivity.

In applications like the test infrastructure - required to handle the "space system" at runtime using the static model definition - a runtime instance of the SSM (RT-SSM) has the role:

- To provide runtime services to SE definition for Act execution and Rdt references with the purpose to monitor and control space system status and behaviour
- To maintain an updated snapshot of the SSM status according to executed activities effects and reporting data samples

Within the verification infrastructure the RT-SSM is implemented through the adoption of a framework and components where the framework represents the components container and components functional add-ons contributing to the RT-SSM dynamic functionality and management.

An RT-SSM component is designed with a *functional layer* implementing the component core specific functionality - and made available in terms of services and data types - and a *semantic adapter* allowing to map the component services and data to the SSM semantic paradigm (SE, Act, Rdt and Evt) supported by the framework.

E-70-32 APPLICABILITY

The ECSS-ST-70-32C identifies the requirements to be satisfied by any language used for the development of automated test and operations procedures. E-70-31 has a level of applicability to E-70-32 as the latter represents the reference language specifications for the implementation of SSM activities of type "procedure". E-70-32 addresses:

- Procedure structure and dynamic behaviour specifications
- Language and semantic specifications of the procedure language
- Syntax of the PLUTO reference language implementing the above specifications

While the specifications and PLUTO have proved to be effective within operational systems, limits have been highlighted as well as a level of usage lessons-learned. The following can be mentioned:

- Testing environments interact with the Space System with a higher level of complexity than Flight Operations environments where procedure are more dedicated to achieve mission operation goals rather than low-level interactions, delegating the low-level handling and complexity of the space system to the SSM.
  Such delegation has proved to be practically not enough for testing where complex data handling may be required together with constructs and features allowing expressing complex procedure goals behaviour and conditions handling.
- As the SSM represents the means for E-70-32 to interact with the space system, the standard miss to address properly all SSM interactions and managements needs.
  The missing specifications are related to aspects such as: support and handling of data of type SSM objects, SSM objects traversing capabilities, searching and query, dynamic SSM objects creation and deletion (dynamic SSM), access-rights, locking, synchronisation

In relation to the latter point, it must be noted that relationship to SSM definition and management has always been a major element of assessment for E-7032 applicability within a specific domain.

Besides the high level areas above, other E-70-32 limitation have been highlighted such as: lack of exception handling, procedure output arguments, capability to express complex conditions criteria.

In order to cope with all mentioned aspects a tailoring exercise can be performed but, especially in the case of the testing domain, this may result in a relevant number of changes and the extensive level of tailoring may imply questioning - or even invalidating the concept of "applicability of a standard".

E-70-32 adoption within the testing infrastructure retains all current standards semantics and runtime specifications with a set of enhancements covering the relevant limitations.

E-70-32 ENHANCEMENTS OVERVIEW

The areas of enhancements can be classified as:

- Formal data model definition covering simple and complex types with extensions capabilities for user defined types. This includes objet references (variables and constants) handling capabilities
- SSM references and management including possibility to define variables and constants of type SSM objects as well as allowing to manage the SSM structure (traversing, search and query)
- Behavioural enhancements through actions handling and annotations
- Other enhancements such as pre-conditions and confirmation bodies rules capabilities, procedure returned arguments , built-in and user defined functions

In terms of language definition the engine supports a grammar expressed with a PLUTO-like syntax as well as XML representation compliant to an XML schema directly derived from the grammar syntax.

The revised syntax is designed as balance between the original PLUTO natural language approach and the need to support the new enhancements avoiding a syntax resulting too verbose.

The XML representation can be considered a formal language definition for interoperability and underlying model for future procedure tabular and graphical editors.

Data Types Model

The current standard does not include a complete formal definition of a data types model as part of the procedure language environment. Simple types and complex types are referenced but not part of a formal data type hierarchy. In additions SSM Objects – SE, Act, Rdt, Evt – are not formalised as supported embedded complex types.

The enhancements include such formalisation and include SSM objects as part of the model complex data types set together with specific space domains types such as CCSDS Source Packet Protocol (SPP) data structures.

The engine allows the user to define custom complex types and make them available at procedure definition and compilation time.

SSM References and Management

The presence of the space system model imposes the capability to handle the SSM definition and references from procedures. The current standard includes only the notion of direct path reference (e.g. "Act1 of SE1.1 of SE1 of SSM") and the support of SSM context (but context definition is still an implicit direct path reference). In other words the standard does not address:

- Indirect references to SSM object, such as usage of local and global variables and constants, used to reference SSM Objects instead of the direct path references. Note that SSM indirect references, besides fulfilling specific needs in support to procedure algorithm definition, is considered an essential features facilitating procedure consistency checking and refactoring (e.g. only a constant or global variable must be redefined)
- SSM traversing and searching capabilities such as it is possible to iterate and query the SSM for SSM structure detection of presence of SSM object and/or handling of their properties

The enhancements cover both capabilities and it must be noted that SSM Object references are a direct consequence of the definition of SSM Objects as part of the supported data model.

In addition the enhancements in this area are in support of the RT-SSM handled as common semantic runtime technology such that language and model integration is fully achieved at runtime.

Behavioural capabilities

Actions

The introduction of "actions" covers the lack of the standard, to support handling of occurrences, during computation, of multiple behaviors, anomalies and exceptions as outcome of a statement execution and associate different operations - handlers - to each of these occurrences.

Actions can be considered a generalization of *exception handling* and it is based on the *try-catch* paradigm extended to any type of behavior occurrences and not to just exceptions.

An actions-block can be associated to any statement. In the case of "initiate and confirm" it replaces the construct *continuation test (*"in case" *confirmation status : confirmation action…),* which defines how to proceed - *continuation action* - based on the activity *confirmation status* after the associated "initiate and confirm" statement is executed.

Annotations

Annotations are a way to extend capabilities of a language with features not directly included in the language grammar/model and they can be used to support a specific applicability domain.

In the verification engines annotations can be used in support to different scenarios and their usage is in support to the procedure pre-compilation/compilation process and at runtime to express additional dynamic information and/or conditional behavior instructions. Possible use-cases are:

- references to externally defined artifacts - through aliases as access keys - and used by the procedure at runtime such as the indirect reference allows to decouple the executable procedure from the artifact definition
- Procedure Code import (static de-referencing), that is, possibility to include procedures codes blocks defined within other sources and imported in the target procedure. This would support the definition of test procedure logic outside the procedure editing and execution environment and import them as part of the finalized executable procedure definition, implicitly allowing decoupling definition of procedure artifacts within an off-line environment (e.g. Pass/fail criteria associated to SUT requirements) and low-level aspects implicitly required by an executable procedure.

Annotations, together with the language capabilities represent a way to tailor and enhance the language engine capabilities without directly acting on the language semantic and grammar.

Annotations are included in annotations blocks (with delimiter tags) have their own syntax and as mentioned, can be extended to cover specific needs.

Other Enhancements

Pre-Conditions - Confirmation Rules

In the current version of the standard pre-condition and confirmation bodies - if present - contains conditional rules (1 or more in terms of "wait" or "if" expression conditions) and each rule "true" or "false" result is combined into a logical AND in order to determine if the whole pre-condition or confirmation is true.

This is not considered sufficient in case complex rules must be expressed and the conditional checks of each rule requires to execute statements as part of the rule "false" and "true" branches.

These have been introduced as enhancements allowing to optionally express rules with the full "if-then-else" construct and allowing to include statements - limited in types - in the respective branches as well nested rules. This, combined with the possibility to include actions and annotations, provides a flexible paradigm to express complex and configurable pre-conditions and confirmation logic.

Procedure output arguments (through procedure properties)

E-70-32 does not formally include the concept of procedure return arguments as a mechanism to return data to an invoking activity. However this can be achieved using activity properties definition with each property used for storing references to data objects.

Standard properties are part of each activity definition and returned arguments can be considered the set of activity custom properties defined with this purpose. This approach has been preferred and formalized instead of inclusion of returned argument definition part of activity definition, as it presents more flexibility and is already – implicitly – covered by E-70-32.

Built-in functions and user-defined functions (custom functions)

Although not specifically affecting language specifications, the language engine extends the type and numbers of built-in functions (string, date-time and math) and includes the possibility to define custom functions in Java and expose them in the procedure language with a language syntax compliant signature. The mechanism obviously allows extending functions capabilities for specific computational and algorithms needs.

OVERVIEW OF USED TECHNOLOGIES

The RT-SSM framework - implemented in java - provides core services for RT-SSM management and components add-on plug-in mechanism. The components provide services covering data definition (e.g. SUT modelling, test plan, etc..), interfaces to external entities (e.g. SUT) and functionality accessible internally to the RT-SSM and to the end user such as procedure language engine and activity executor.

OSGi (and in particular Eclipse Equinox) has been adopted for framework specifications where each component is implemented as bundle (or set of bundles). OSGi [3] has been selected due to a number of essential benefits but mainly is in line with the fundamental requirement to implement functionality and add-on according to formal component design specifications instead of design guidelines.

The framework can be instantiated within a standalone application (e.g. coupled with the user applications layer) as well as in a client-server configuration with a possible future distributed configuration (distributed RT-SSM) using messaging bus technology.

In terms of language, modelling and artefacts generation, Eclipse Xtext [4] and EMF constitutes the fundamental adopted technologies. Xtext provides an environment allowing defining domain specific languages with generation of runtime components (e.g. parser, compiler skeleton, abstract syntax tree) integrated with and based on the Eclipse Modelling Framework (EMF). The model of the language grammar is maintained by and EMF model through which the language XML schema is generated. In addition Xtext + EMF environment generates components in support to language IDE with features like syntax colouring and code editing content assistant.

CONCLUSIONS

The Space System Model (static and runtime) and E-70-32 applicability and language extensions constitutes the core conceptual and implementation elements of the testing infrastructure.

The formalisation of SSM as common semantic model mapped to the RT-SSM with a design based on framework and components provides the infrastructure with a formal yet flexible approach for the infrastructure functional deployment with a high level and flexible customisation capabilities.

The E-70-32 specifications and the identified enhancements are believed to provide a powerful level of constructs and features in support to a wide range of automatic verification test scenarios.

On the standards applicability side the described infrastructure has retained all fundamental E-70-31 and E-70-32 concepts and specifications

In relation to E-70-32, the enhancements may eventually contribute to the definition of a future revised version as part of standard assessment and evolution. Such activity could take into account:

- Lessons learned from E-70-32 applicability within existing systems as well as level of tailoring defined within specific domains, like the described verification infrastructure
- The set of features can be expressed as grammar but as well as XML schema such that procedure definition and validation can rely on the schema and not on a parser technology
- The standard features and customisation mechanisms may be used as reference on top of which higher Domain Specific Languages can be defined.

  DSLs can be used to define an environment in support to a specific problem domain to be managed by an entity having a specific responsibility in the Space System definition, testing and operation.

  Compatibility and interoperability would always be achieved through the standard specifications (grammar and XML schema). Present technologies and tools are mature to support such approach.

REFERENCES

[1] ECSS-E-ST-70-31C - Space Engineering *Ground systems and operations - Monitoring and control data definition*. 31 July 2008[E-70-31] ECSS-E-ST-70-31C - Space Engineering *Ground systems and operations - Monitoring and control data definition*. 31 July 2008

[2] ECSS-E-ST-70-32C - Space Engineering *Ground systems and operations - Test and operations procedure language*. 31 July 2008[E-70-31] ECSS-E-ST-70-31C - Space Engineering *Ground systems and operations - Monitoring and control data definition*. 31 July 2008

[3] OSGi - www.osgi.org

[4] Eclipse Xtext - http://www.eclipse.org/Xtext/