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

Francesco Croce
EUMETSAT
francesco.croce@eumetsat.int

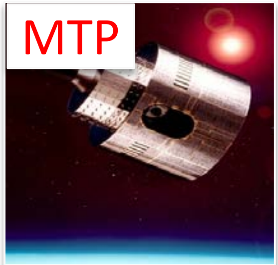
EUMETSAT

Agenda

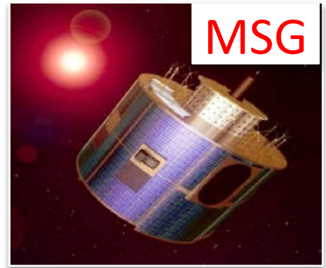
- Background Information
- ECSS-E-70-31 Role and Adoption
- ECSS-E-70-32 Current Version Assessment
  Language and Engine Extensions
  Language Extensions Examples
- Overview of Adopted Technologies
- Conclusions

Missions Under EUMETSAT Flight Operations Responsibility

Geostationary



- **MTP-1**



- **MSG-1**
- **MSG-2**
- **MSG-3**
- *MSG-4 (Q1-2015)*

Under Definition or Development

- *MTG-I1*
- *MTG-I2*
- *MTG-I3*
- *MTG-I4*

- *MTG-S1*
- *MTG-S2*

Polar

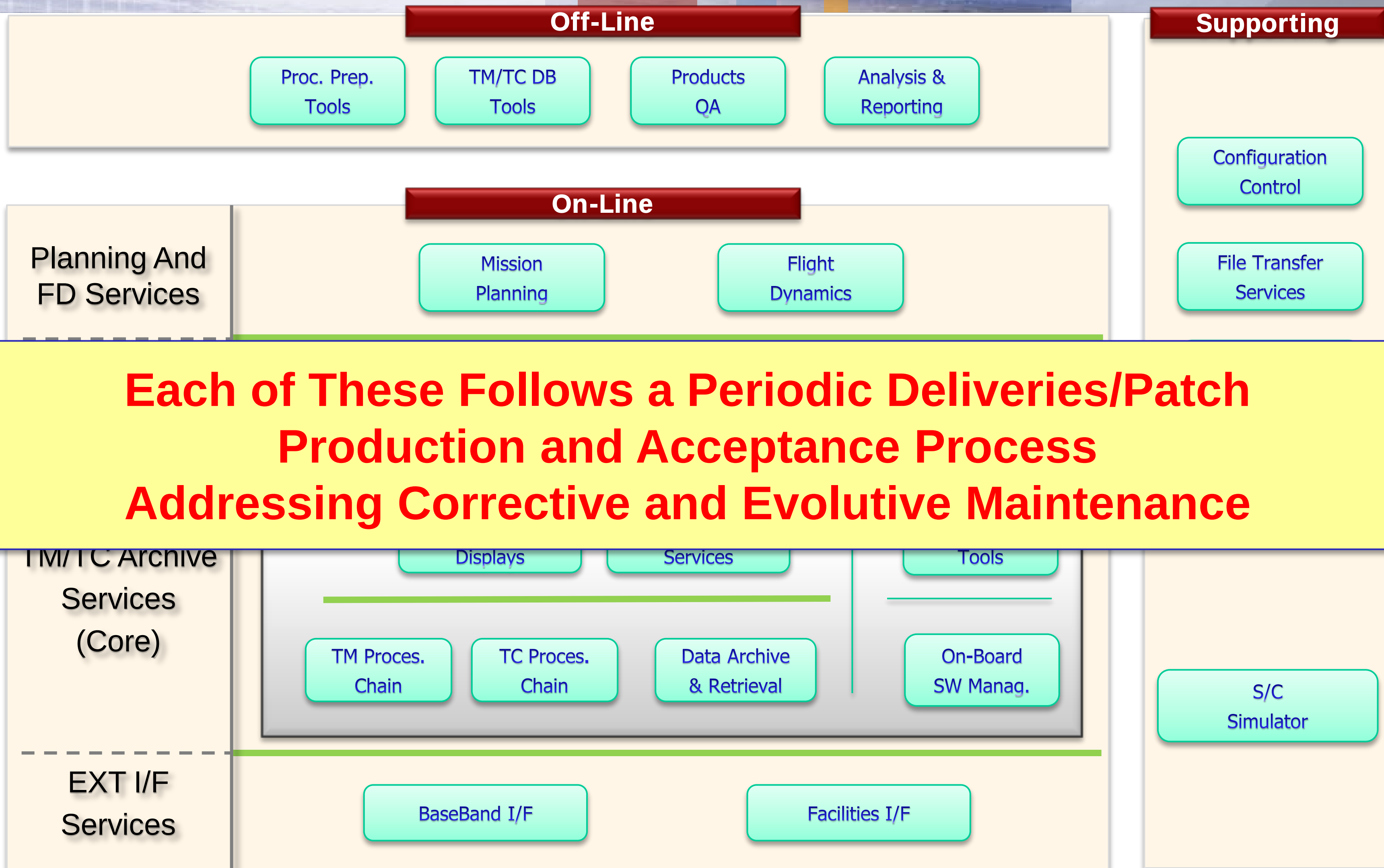
- **METOP-A**
- **METOP-B**
- *METOP-C (2017)*

Under Definition or Development

- *Sentinel-3A*
- *Sentinel-3B*

- *EPS-SG 1A / 1B*
- *EPS-SG 2A / 2B*
- *EPS-SG 3A / 3B*

Mission Control System Applications/Tools Functional Domain



Each of These Follows a Periodic Deliveries/Patch Production and Acceptance Process Addressing Corrective and Evolutive Maintenance

M&C Applications
Verification Infrastructure Initiative

The M&C Applications Maintenance and Engineering Team is defining and implementing a generic MCS applications **testing and verification infrastructure**

High level goals:

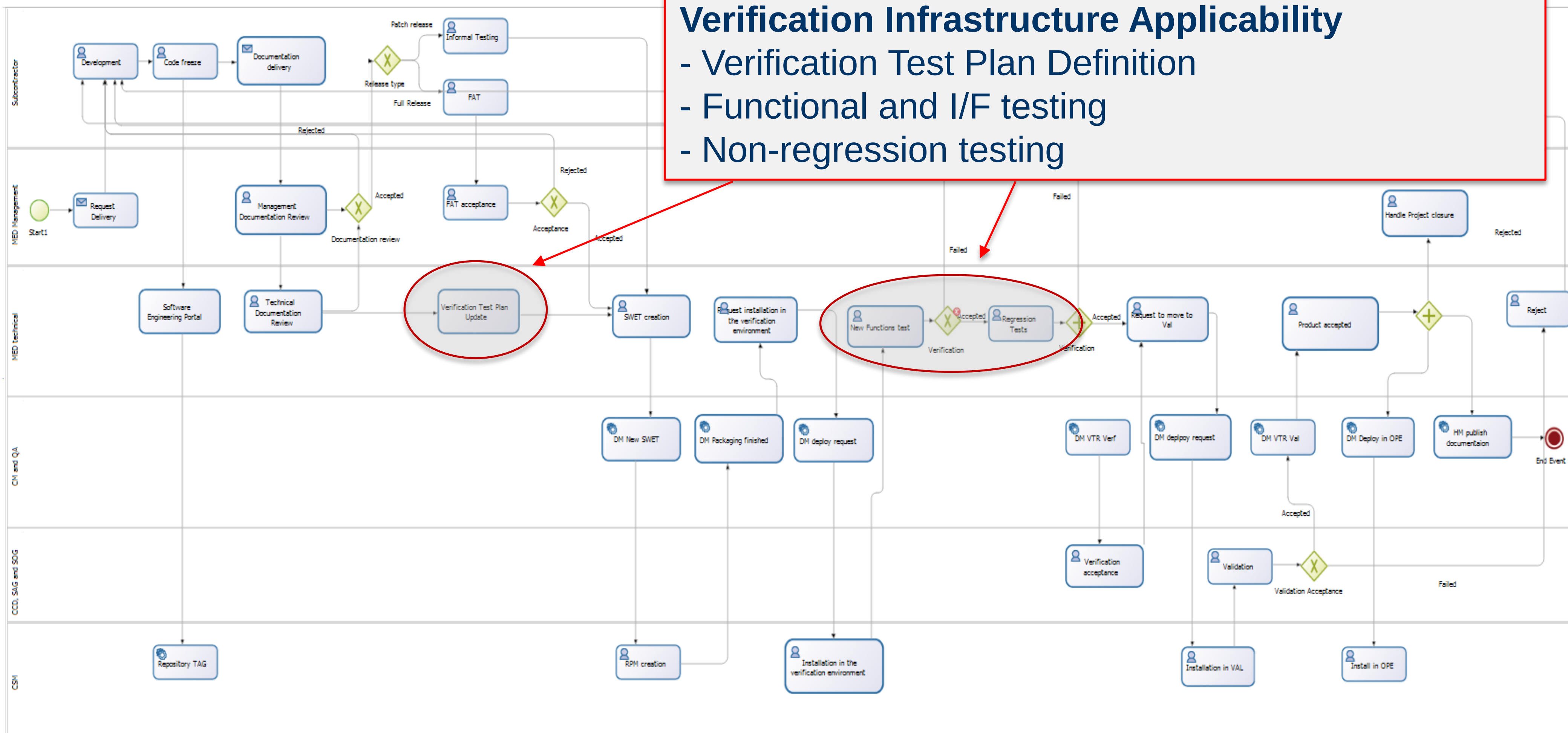
- To **streamline** and **harmonize** the verification process through a common infrastructure and test artefacts model

- Testing **automation** in terms of:
  - o execution with **test pass/fail assessment and closed loop reference** with the formal System Under Test (**SUT**) requirements (e.g. SRD, SRS)
  - o Documentation production (e.g. Test Results)

Automation is fundamental in support to a fast and yet formal **regressions testing** within a verification process of a new SUT patch/delivery

MCS Applications Delivery/Patches Production, Verification and Roll-Out Process

Verification Infrastructure Applicability
- Verification Test Plan Definition
- Functional and I/F testing
- Non-regression testing



EUMETSAT

Infrastructure
Fundamental Required Features and Capabilities

Extensible Data-Model

- **Formal data-model definition and handling**

  Covering simple and complex data types with possibility of extensions to custom types (user defined)

Flexible Ext I/F Customization

- **SUT I/F and Supporting tools Customization**

  Need to support different: **SUT interfaces** (technologies, mechanisms, ICDs): interfaces to **emulator/simulators**, interfaces to external tools used for test definition and requirements management (i.e. DOORS)

Components Based Technology

- **Adoption of Formal Components Based Technology**

  allowing extensions though components-based approach ruled by formal specifications

Core Components and Standards Adoption

Space System Model

as common semantic and runtime technology

ECSS E-ST-70-31C inspired

Test procedure language definition and execution

ECSS-E-ST-70-32C

With Extensions

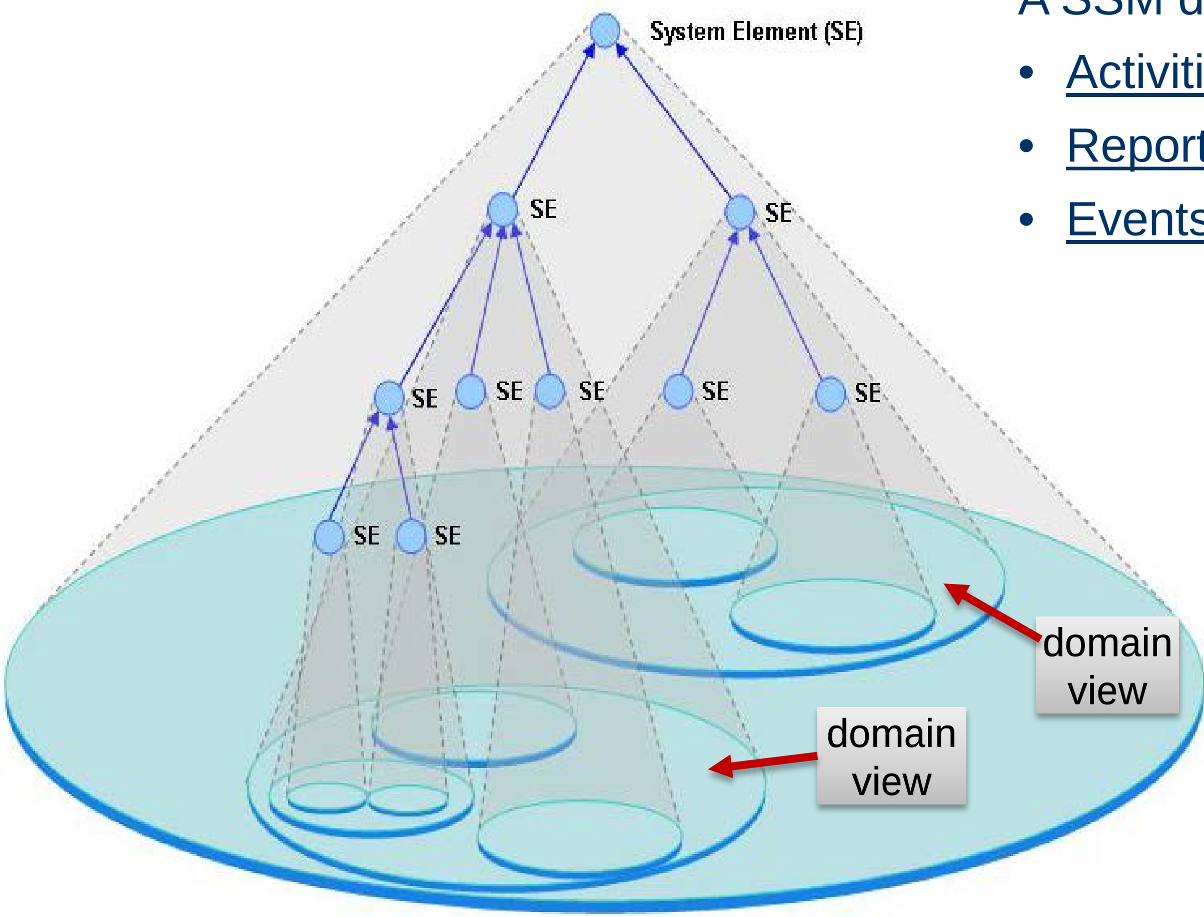
Formal Data Model

EUMETSAT

Space System Model
Role and Adoption

Space System Model Concepts

The SSM captures the Space System information and knowledge in terms of functional and/or physical hierarchy of System Elements (**SE**)



A SSM defines each SE knowledge in terms of :

- Activities (**Act**)
- Reporting Data (**Rdt**)
- Events (**Evt**)

A functional/physical entity may be modeled by a **domain-specific view** (or sub-views)

Each view/sub-view modeling:

- the particular domain of interest of the entity
- specific functional application and/or behavior associated to the entity.

EUMETSAT

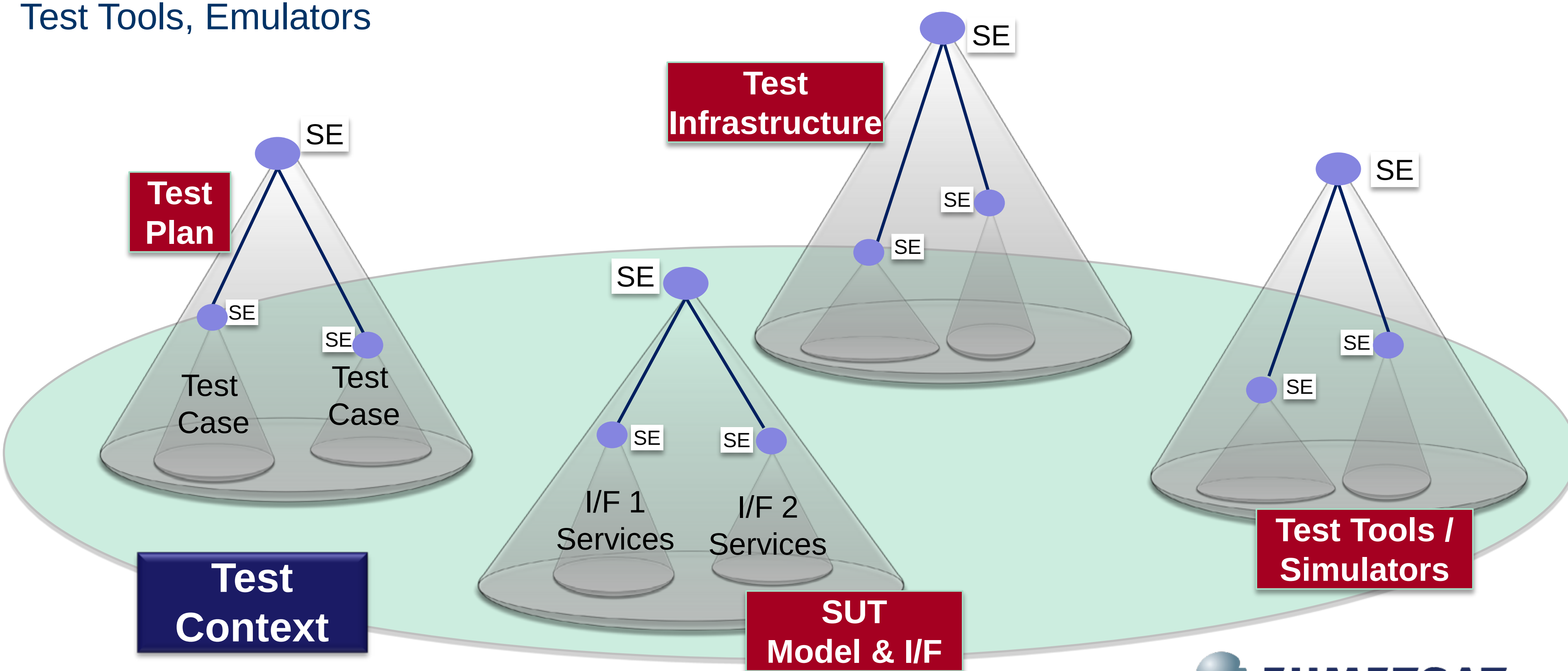
SSM as Common Semantic

The concepts of SE hierarchy, Act, Rdt, Evt and domain-specifc views are considered a generic **semantic** that can be used to model elements of a specific SSM applicability domain

The adoption of common semantic as allows to rely on an abstract layer above low-level technologies, decoupling technologies from meaning

SSM Common Semantic In the Test Context

- Test Plan (as **SE**), Test Cases (as **SE**), Test Procedures (as **Act**)
- SUT interfaces, services and data (hierarchy of **SE** and associated **Act**, **Rdt**, **Evt**)
- Test infrastructure functionality like activity executor, ….
- Test Tools, Emulators



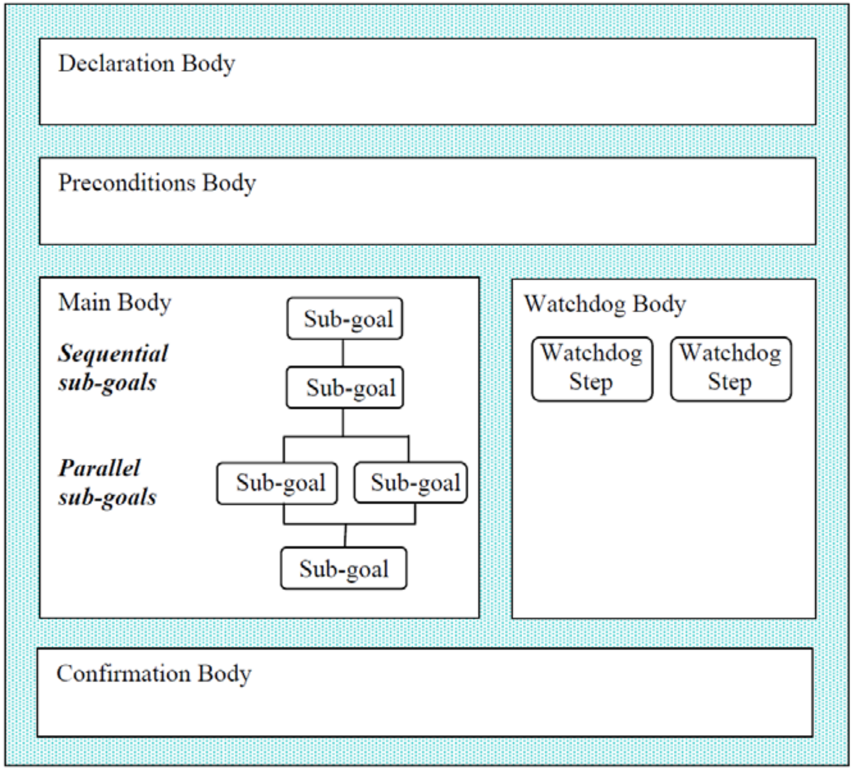
E-70-32
Adoption and Assessment

ECSS-E-70-32C

ECSS-ST-70-32C identifies the requirements to be satisfied by any language used for the development of *automated* **test** and **flight** operations procedures

The standard addresses:

- Procedure **structure** and **dynamic behavior** specifications
- Procedure Language **semantic** specifications
- Syntax of the **PLUTO reference language** implementing the specifications



The testing infrastructure adopts all current standard specifications

E-70-32 Limitations – 1
Testing Environment vs. Flight Operations

Flight Operations Procedures tend to be simple and dedicated to achieve mission operation goals, somehow **delegating** the low-level handling of the Space System complexity to external entities (hence to the **SSM** in E-70-32)

Testing environments interact with the Space System typically with a **higher level of complexity** than FOPs

In testing the delegation to the SSM is considered not enough for testing, and the standard misses a level of formalization for:

- **complex data types definition, handling and manipulation**
- Availability of flexible **semantic constructs/features** required to express complex pre-conditions, testing goals, behavior, conditions handling (including exception handling) and confirmation criteria

E-70-32 Limitations – 2
Handling, Interaction and Management of SSM References

The standard does not address properly all SSM interactions and SSM managements needs such as:

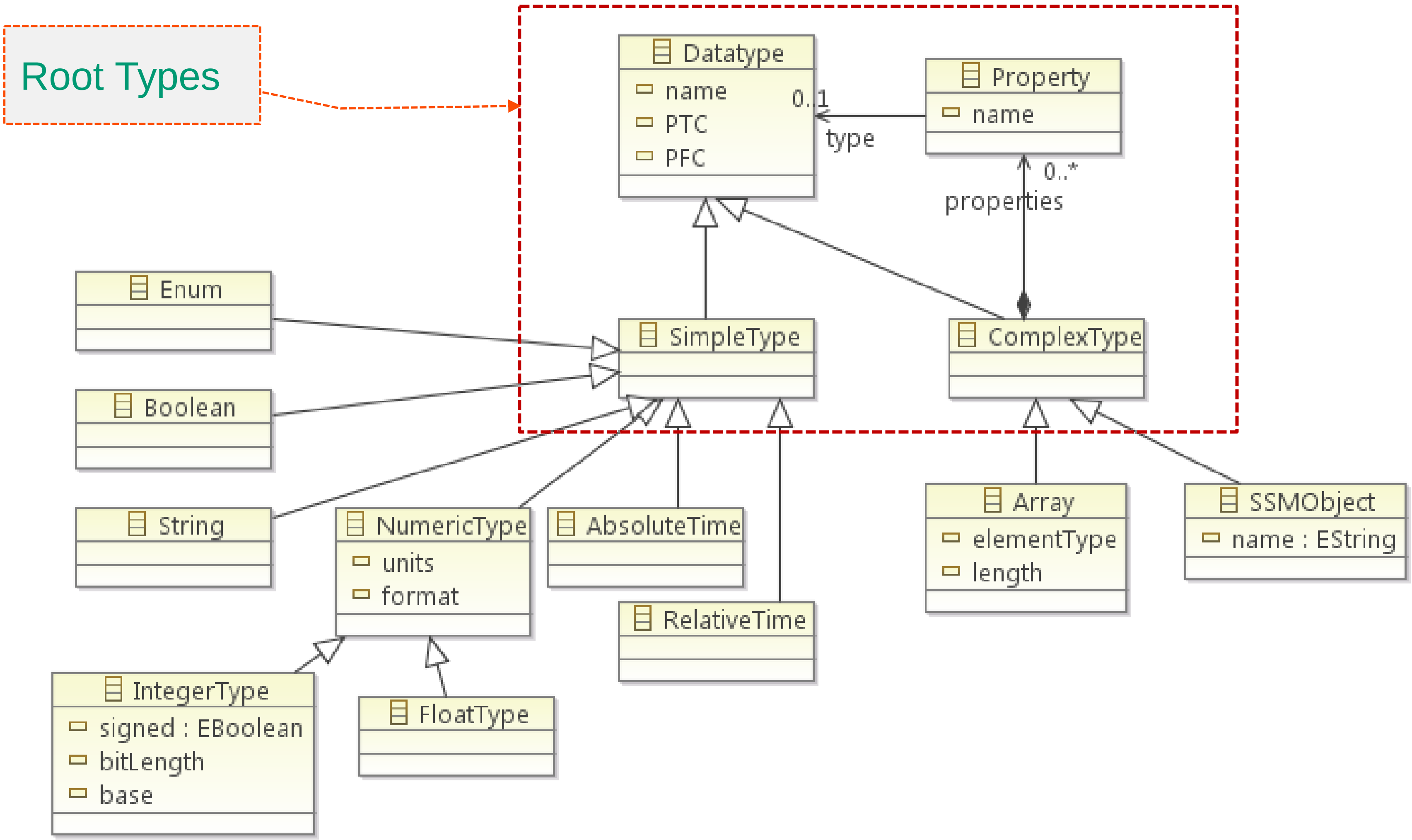
- Declaration and handling of **SSM-Object** data types (SE, Act, Rdt, Evt) and **SSM-Object References**
- SSM structure traversing as well as SSM Objects properties searching and query
- Dynamic SSM (Dynamic SSM) management with SSM-Objects creation, deletion, etc..
- Access-rights, locking, synchronization

EUMETSAT

E-70-32

Language and Engine Extensions

Data Model Definition
Simple Type, Complex Types, Properties

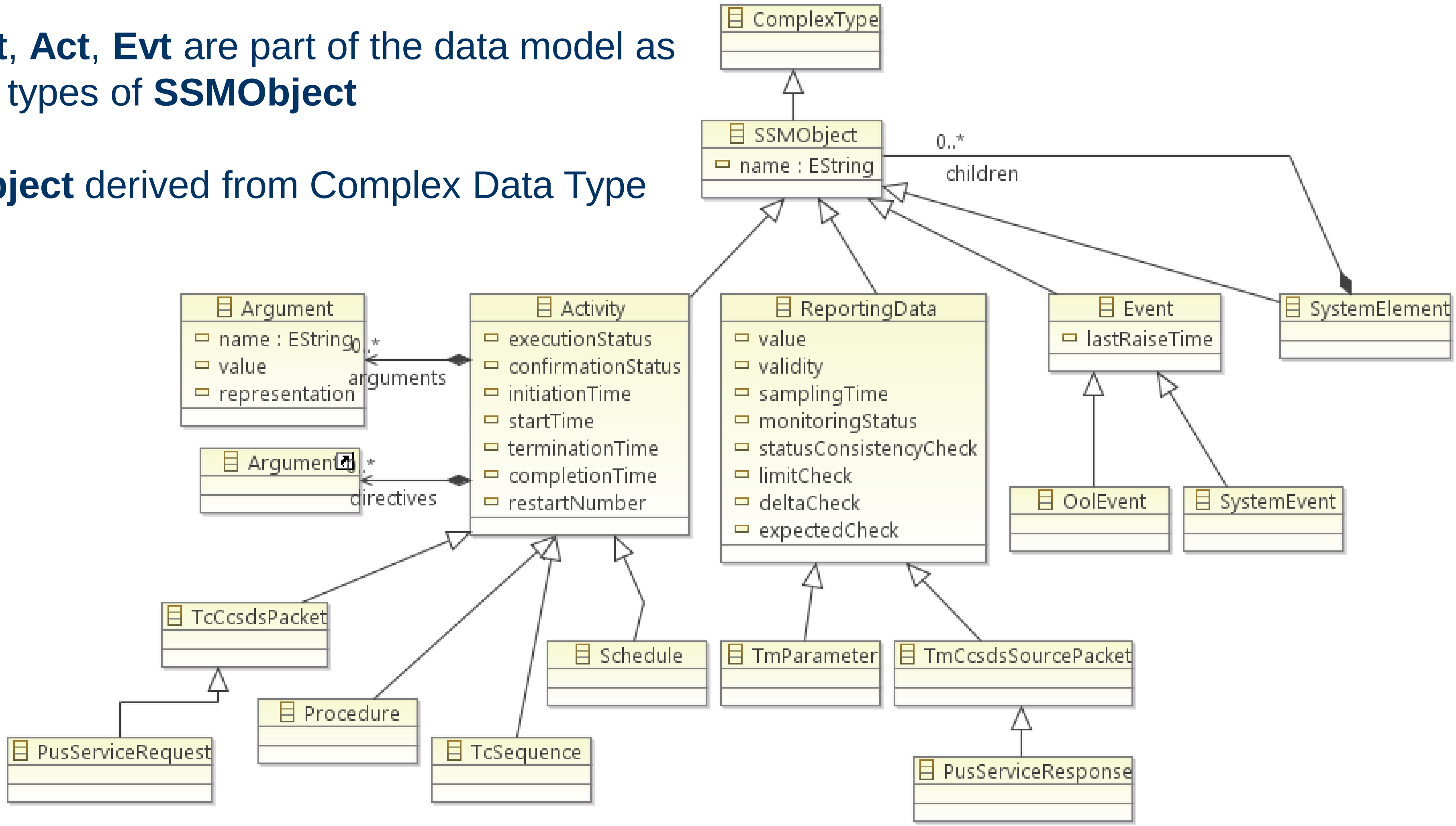


EUMETSAT

Data Model Definition
SSM Objects as Complex Data Type

SE, **Rdt**, **Act**, **Evt** are part of the data model as derived types of **SSMObject**

SSMObject derived from Complex Data Type



Language Extensions Overview - 1

Support to Formal data model definition

- simple and complex data types including SSMObjects (SE, Act, Rdt, Evt)
- extensions capabilities mechanism for custom-types (user defined)

SSM references and management

- possibility to define variables and constants of type SSM objects (*SystemElement*, *Activty*, *ReportingData*, *Event*) and SSM Object references
- SSM structure and SSMObjects traversing, search and query capabilities as part of **standard methods** associated to SSMObjects Data Types

Language Extensions Overview - 2

Behavioral Enhancements

- actions handling (generalization of exception handling)
- Enhanced pre-conditions and confirmation rules capabilities
- Annotations as a way to extend capabilities of the language with additional features not directly included in the language grammar/model

Other Enhancements

- procedure returned arguments
- Enhanced built-in functions
- Capability to define user custom functions

Language Definition

In terms of language definition the language grammar supports:

- a EBNF syntax expressed as PLUTO-like
- **XML schema** directly derived from the syntax

The syntax is designed as a balance between:

- the original PLUTO natural language approach
- The need to support the new features with an optimized approach (i.e. avoiding unnecessary “verbose” constructs)

E-70-32

Language Extension Examples

Object References

Procedure

```
testPlan:        reference(SystemElement);
testCaseList:    reference(SystemElement)[];

step ExecuteTestPlan

    TestPlan = ssm.find ( "MainTestPlan",
                          SE_TYPE_TESTPLAN,
                          SSMOBJ_CLASS_SYEL);

    testCaseList = TestPlan.getChildren (SE_TYPE_TESTCASE,
                                         SSMOBJ_CLASS_SYEL,
                                         LEVEL_ALL);

end step ExecuteTestPlan
end procedure
```

testPlan variable of type *SystemElement* Reference

testCaseList as variable of type Array of *SystemElement* Reference

Find in the ***ssm*** objects reference pointing to a SE of type TESTPLAN with name "MainTestPlan"

ssm is a global object reference pointing to the SSM root

Get all children SEs of type TESTCASE contained by ***TestPlan***

Actions: Generalization of Exception-Handling

Procedure

```
testPlan:        reference(SystemElement);
testCaseList:    reference(SystemElement)[];

step ExecuteTestPlan
    actions {
        TestPlan_SE = ssm.find ("MainTestPlan", SE_TYPE_TESTPLAN,
                                SSMOBJ_CLASS_SYEL);
    } handle ObjectReferenceNotFound {
        print("Test Plan Not Found");
        terminate();
    }
....
end step ExecuteTestPlan
end procedure
```

Actions Block with **handlers** for each condition

EUMETSAT

Initiate and Confirm with Actions

```
…
      testProcedureList:    reference(SystemElement) [];
       testProcedureList = TestCase.getChildren (….);
 …
      for (tprocIdx=0; tprocIdx < totalTestProcedures; tprocIdx++) {

         initiate and confirm testProcedureList[tprocIdx]
         handle Confirmed {
            print(testProcedureList[procIdx].name + "confirmed.");
            totalConfirmedTestProcedures++;

         } handle NotConfirmed {
            totalNotConfirmedTestProcedures++;
            print(testProcedureList[procIdx].name + " not confirmed.");
            print("Reason:“ + testProcedureList[procIdx].messages);

         }
….
```

Actions Block for initiate and confirm

Initiate and Confirm with Actions

…

confirmation

Confirmation Body with multiple complex rules with if-then-else brances
possibility to include statements on each conditional branch
The same capability is available for pre-conditions

```
if ( totalConfirmedTestProcedures == totalTestProcedures ) {
        print ("Test Plan Successfully Executed");

} else {
        print ("Test Plan With Failed Procedures (" +
                totalNotConfirmedTestProcedures + " failed  out of " +
                totalTestProcedures);
}
```

end confirmation

…

Brief Overview of Adopted Technologies

Technologies

RT-SSM framework and RT-SSM Components (e.g. SUT I/F, test plan DB definition, etc..)

- **java**
- **OSGi** (Eclipse Equinox) And **Spring** (springsource)
- components as formal OSGi **Bundles**

OSGi 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

Language, modeling and supporting artifacts (parser, editor)

- Eclipse **Xtext** and **EMF**

Messaging and Integration

- **ApacheMQ**
- **Apache Camel**

Conclusions

Conclusions - 1

- The verification infrastructure has retained all fundamental E-70-32 specifications and the adoption of the E-70-31 SSM Concepts
- The formalization of SSM as semantic model provides a formal and generic paradigm to model all elements and entities in test domain
- The E-70-32 specifications and the identified extensions are believed to provide a of test language constructs and features in support to a wide range of automatic verification test scenarios and complexity

Conclusions - 2

- The E-70-32 extensions may contribute to the definition of a future revised version of the standard. Such E-70-32 update 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 (e.g. the described verification infrastructure)
- Language features can be defined as grammar syntax and as XML schema, with XML used for procedures interoperability
- The language features may be used as reference specifications on top of which higher **Domain Specific Languages (DSL)** can be defined

  DSLs built on-top of the E-70-32 can be used to define a language environment in support to a specific problem domain within the space system, but at the same time….

  …maintaining **compatibility** and **strong interoperability** with the E-70-32 formal specifications (grammar and XML schema)

  Present technologies and tools are mature to support DSL definitions

Thank You