ECSS-E-TM-40-07 Volume 1A
25 January 2011

Space engineering

Simulation modelling platform - Volume 1: Principles and requirements

ECSS Secretariat
ESA-ESTEC
Requirements & Standards Division
Noordwijk, The Netherlands

Foreword

This document is one of the series of ECSS Technical Memoranda. Its Technical Memorandum status indicates that it is a non-normative document providing useful information to the space systems developers' community on a specific subject. It is made available to record and present non-normative data, which are not relevant for a Standard or a Handbook**.** Note that these data are non-normative even if expressed in the language normally used for requirements.

Therefore, a Technical Memorandum is not considered by ECSS as suitable for direct use in Invitation To Tender (ITT) or business agreements for space systems development.

Disclaimer

Published by: ESA Requirements and Standards Division
ESTEC, P.O. Box 299,
2200 AG Noordwijk
The Netherlands

Change log

ECSS-E-TM-40-07 Volume 1A 25 January 20111	First issue

Table of contents

Tables

Introduction

Space programmes have developed simulation software for a number of years, and which are used for a variety of applications including analysis, engineering operations preparation and training. Typically different departments perform developments of these simulators, running on several different platforms and using different computer languages. A variety of subcontractors are involved in these projects and as a result a wide range of simulation models are often developed. This Technical Memorandum addresses the issues related to portability and reuse of simulation models. It builds on the work performed by ESA in the development of the Simulator Portability Standards SMP1 and SMP2.

This Technical Memorandum is complementary to ECSS-E-ST-40 because it provides the additional requirements which are specific to the development of simulation software. The formulation of this Technical Memorandum takes into account the Simulation Model Portability specification version 1.2. This Technical Memorandum has been prepared by the ECSS-E-40-07 Working Group.

This Technical Memorandum comprises of a number of volumes.

The intended readership of Volume 1 of this Technical Memorandum are the simulator software customer and all suppliers.

The intended readership of Volume 2, 3 and 4 of this Technical Memorandum is the Infrastructure Supplier.

The intended readership of Volume 5 of this Technical Memorandum is the simulator developer.

- **Volume 1 – Principles and requirements**

  This document describes the Simulation Modelling Platform (**SMP**) and the special principles applicable to simulation software. It provides an interpretation of the ECSS-E-ST-40 requirements for simulation software, with additional specific provisions.

- **Volume 2 - Metamodel**

  This document describes the Simulation Model Definition Language (**SMDL**), which provides platform independent mechanisms to design models (Catalogue), integrate model instances (Assembly), and schedule them (Schedule). SMDL supports design and integration techniques for class-based, interface-based, component-based, event-based modelling and dataflow-based modelling.

- **Volume 3 - Component Model**

  This document provides a platform independent definition of the components used within an SMP simulation, where components include models and services, but also the simulator itself. A set of mandatory interfaces that every model has to implement is defined by the document, and a number of optional interfaces for advanced component mechanisms are specified.

  Additionally, this document includes a chapter on Simulation Services. Services are components that the models can use to interact with a Simulation Environment. SMP defines interfaces for mandatory services that every SMP compliant simulation environment must provide.

- **Volume 4 - C++ Mapping**

  This document provides a mapping of the platform independent models (Metamodel, Component Model and Simulation Services) to the ANSI/ISO C++ target platform. Further platform mappings are foreseen for the future.

  The intended readership of this document is the simulator software customer and supplier. The software simulator customer is in charge of producing the project Invitation to Tender (ITT) with the Statement of Work (SOW) of the simulator software. The customer identifies the simulation needs, in terms of policy, lifecycle and programmatic and technical requirements. It may also provide initial models as inputs for the modelling activities. The supplier can take one or more of the following roles:

  — Infrastructure Supplier - is responsible for the development of generic infrastructure or for the adaptation of an infrastructure to the specific needs of a project. In the context of a space programme, the involvement of Infrastructure Supplier team(s) may not be required if all required simulators are based on full re-use of exiting infrastructure(s), or where the infrastructure has open interfaces allowing adaptations to be made by the Simulator Integrator.

  — Model Supplier - is responsible for the development of project specific models or for the adaptation of generic models to the specific needs of a project or project phase.

  — Simulator Integrator – has the function of integrating the models into a simulation infrastructure in order to provide a full system simulation with the appropriate services for the user (e.g. system engineer) and interfaces to other systems.

- **Volume 5 – SMP usage**

  This document provides a user-oriented description of the general concepts behind the SMP documents Volume 1 to 4, and provides instructions for the accomplishment of the main tasks involved in model and simulator development using SMP.

1
Scope

ECSS-E-TM-40-07 is a technical memorandum based on ECSS-E-ST-40 for the engineering of simulation software.

It includes:

a. the interpretation of the ECSS-E-ST-40 requirements for simulation software, with additional specific provisions;

b. the tailoring of some provisions of the ECSS-E-ST-40 requirements for simulation software;

c. special practices applicable for simulation software;.

ECSS-E-TM-40-07 follows the structure of ECSS-E-ST-40.

ECSS-E-TM-40-07 complements ECSS-E-ST-40 in being more specific to simulator software, it indicates new provisions, ways of tailoring, and common practices in the domain, but it does not attempt to give detailed descriptions of how to carry out the processes defined in ECSS-E-ST-40. It indicates, however, particular practices that can be applicable for simulation software engineering.

This technical memorandum can be used:

a. as complement to ECSS-E-ST-40 for the additional requirements which are specific to the development of simulation software.

b. to help customers in using and tailoring ECSS-E-ST-40. This document provides the additional requirements which a customer can make applicable on the simulator software developed by the supplier.

c. to assist suppliers in using ECSS-E-ST-40. This document provides a technical specification which a supplier must follow in-order to be compliant with the requirements specific to the development of simulation software.

d. to assist customers and suppliers in adapting or writing organizational procedures and standards that conform to the requirements of ECSS-E-ST-40.

In order to help the users easily identify the normative provisions, they are provided in separate sub-clauses.

This Technical Memorandum may be tailored for the specific characteristic and constrains of a space project in conformance with ECSS-S-ST-00.

2
Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this ECSS Technical Memorandum . For dated references, subsequent amendments to, or revision of any of these publications do not apply, However, parties to agreements based on this ECSS Standard are encouraged to investigate the possibility of applying the more recent editions of the normative documents indicated below. For undated references, the latest edition of the publication referred to applies.

ECSS-S-ST-00-01	ECSS system — Glossary of terms
ECSS-E-ST-40	Space engineering – Software

3
Terms, definitions and abbreviated terms

3.1 Terms from other standards

For the purpose of this Technical Memorandum, the terms and definitions from ECSS-ST-00-01 and ECSS-E-ST-40 apply.

3.2 Terms specific to the present technical memorandum

3.2.1 aggregation, aggregate

A component may aggregate functionality of other components via interface references. The component then acts as a consumer of functionality that other components provide via interfaces.

Component that aggregates functionality of other components via interface references

NOTE The aggregate then acts as a consumer of functionality that other components provide via interfaces.

Contrast: composition, composite

See: component, provider, consumer, interface, reference

UML: aggregation, component, provided interface, required interface

3.2.2 assembly

An assembly describes the configuration of a set of model instances, specifying initial values for all fields, actual child instances, and type-based bindings of interfaces, events, and data fields. Each model instance in an assembly is bound to a specific model in the catalogue for its specification.

NOTE 1 An assembly is independent from the actual model implementations. The actual binding of the model instances of the assembly to the platform specific model implementations is done via the implementation attribute of the model instances. Therefore, an assembly specifies a runtime configuration that can be implemented by many sets of model implementations.

NOTE 2 Assemblies are intended to support a convenient and productive way of re-using or substituting different implementations based on the same simulation architecture.

Contrast: [run-time] configuration

See: model instance, dynamically configured simulation

3.2.3 catalogue

A catalogue contains namespaces as a primary ordering mechanism, where each namespace may contain types. These types can be language types, which include model types as well as other types like structures, classes, and interfaces. Additional types that can be defined are event types for inter-model events, and attribute types for typed metadata. The language for describing entities in the catalogue is the SMP Metamodel, or SMDL.

See: model type

3.2.4 component

A component is a unit of functionality that can be deployed, and that has a well-defined interface to its environment (also called a contract). In SMP, typical components are models and services.

See: model, service, interface, contract

UML: component, provided interface, required interface

3.2.5 composition, composite

A composite component may have other components as children that are held in containers.

Contrast: aggregation, aggregate

See: component, container

UML: component, composition

3.2.6 configuration

A configuration document allows specifying arbitrary field values of component instances in the simulation hierarchy. It can be used to initialise or reinitialise dedicated field values of components in a simulation hierarchy.

3.2.7 consumer [component]

A consumer is a component that either

- invokes operations or properties on another component (called the provider) via a reference to one of the interfaces that the other component provides, or that
- receives event notifications sent from an event source of another component (called the provider) to one of its event sinks, or that
- receives data from an output field of another component (called the provider or [data] source) to one of its input fields.

Contrast: provider

See: event sink, input field, reference

3.2.8 container

A composite component may have other components as children that are held in containers. A container is typed by either an interface or a model type, specifying which types of children may be held in the container. Furthermore, it may specify the minimum and maximum number of allowed children.

Contrast: reference

See: component, composition

UML: component, composition

3.2.9 contract

A contract defines how a component interacts with its environment. Typically, a contract consists of the interfaces provided by the component as well as the references that the component has to other components' interfaces. Furthermore, the contract may also include event sources and sinks.

In another context, a model type may also be seen as a contract for the corresponding model implementation.

See: interface, reference, event source, event sink, model type, model implementation

UML: component, provided interface, required interface

3.2.10 cynamically configured simulation, dynamic configuration term

A dynamically configured simulation is configured from an external file (typically an SMDL assembly file), i.e. model instances are created, configured and connected according to the information therein. In a dynamically configured simulation, a change of the model hierarchy, of initial values, of model links, or of model scheduling can be done without any recompilation by changing the assembly file and reloading the simulation.

Contrast: statically configured simulation

3.2.11 entry point

An entry point is an operation without parameters that does not return a value (so-called void-void operation) that can be added to the Scheduler or Event Manager service, or to a Task that then may be scheduled. In SMP, entry points are executed via an interface rather than via a direct function call as in SMP1/SMI.

In the case of a dynamically configured simulation, a component holding entry points is called a publisher.

See: component, publisher, service

3.2.12 event sink

A component may use an event sink in order to receive a notification when a certain condition in another component is met. The binding to the event source of the other component (called the provider) is done via a subscription mechanism.

Contrast: event source

See: consumer

3.2.13 event source

A component may provide an event source in order to allow other components to subscribe. When a certain condition in the component is met, the component may emit this event and notify all subscribed event sinks held in other components (called the consumers).

Contrast: event sink

See: provider

3.2.14 exception

An exception is a special kind of signal, typically used to signal fault situations. The sender of the exception aborts execution and execution resumes with the receiver of the exception, which may be the sender itself. The receiver of an exception is determined implicitly by the interaction sequence during execution; it is not explicitly specified.

UML: exception

3.2.15 field

A field is a valued feature of a structure, class, or model that is typed by a value type (called data item in SMP1/SMI). Fields typically hold the internal state of a model. They may be published to the simulation environment for the purpose of storing/restoring the state vector (State=true) or for visualisation or data monitoring purposes (View=true).

See: input field, output field, property

UML: attribute

3.2.16 implementation inheritance

Implementation inheritance denotes the inheritance of the implementation of a more general element. Includes inheritance of the interface.

Contrast: interface inheritance

UML: implementation inheritance

3.2.17 input field

In a dataflow-based design, a model may expose input and output fields that are linked to entry points. Exposing an input field indicates that the model requires this field to be updated prior to the execution of the associated entry point.

Note that SMP only standardises this meta-information (in the catalogue) together with the possibility to specify field links (in the assembly). It does not, however, standardise the way in which this information is used in an implementation in order to actually perform the data transfer.

Contrast: output field

See: consumer, entry point, target

3.2.18 interface

Named set of properties and operations that characterize the behaviour of a component or parts of it.

NOTE A component can provide one or more interfaces that can be consumed by other components.

See: aggregation, component, consumer, provider

UML: interface, provided interface

3.2.19 interface inheritance

Interface inheritance denotes the inheritance of the interface of a more general element. Does not include inheritance of the implementation.

Contrast: implementation inheritance

UML: interface inheritance

3.2.20 model [type]

A model [type] specifies a contract for a set of model implementations. It does not depend on a specific target platform. Each model implementation has to implement all artefacts in the contract given by the model type, but it may also extend the contract by adding specific elements like operations, properties, and fields.

Contrast: model implementation, model instance

3.2.21 model implementation

A model implementation implements the functionality of a model that has been defined in a model catalogue. There can be many different implementations of a model defined in a catalogue. The model implementation depends on the target platform (i.e. C++, Java, etc).

NOTE Currently, we do not envisage cross-platform interoperability (i.e. interoperability of model implementations between different platforms). However, in the future, cross-platform bridges may be specified for this purpose.

Contrast: model instance, model type, run-time instance

3.2.22 model instance

A model instance is part of an assembly. It is linked to a model type (specifying the contract) and may optionally specify a model implementation. It serves as a placeholder for the run-time model instance that is created when the assembly is used to instantiate a configuration.

See: assembly

Contrast: model implementation, model type, run-time instance

3.2.23 model library

A model library contains model implementations, usually in binary or compiled form. In terms of implementation in ANSI/ISO C++, the analogous of a model library is a class library.

Note: On Windows platforms, binary class libraries are usually held in Dynamic Link Libraries (DLL), and on UNIX platforms they are usually held in Dynamic Shared Objects (DSO).

Contrast: catalogue

UML: model library

3.2.24 output field

In a dataflow-based design, a model may expose input and output fields that are linked to entry points. Exposing an output field indicates that the model updates this field within the execution of the associated entry point – based on the values of the associated input fields (and possibly some internal state).

Note: SMP only standardises this meta-information (in the catalogue) together with the possibility to specify field links (in the assembly). It does not, however, standardise the way in which this information is used in an implementation in order to actually perform the data transfer.

Contrast: input field

See: entry point, provider, source

3.2.25 operation

An operation is a feature of a class, interface, or model, which declares a service that can be performed by instances. It may have parameters and it may return a value.

An operation may be static, in which case it is also called a class method in OO languages.

Synonym: [instance] method, member function

UML: operation

3.2.26 platform

A platform is a set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns that any subsystem that depends on the platform can use without concern for the details of how the functionality provided by the platform is implemented.

In SMP, typical platforms are: ANSI/ISO C++ with the appropriate component model specified by SMP, CORBA with an appropriate component model like the CORBA Component Model (CCM), Java 2 Enterprise Edition (J2EE) or Java Beans, or the Microsoft .NET platform.

MDA: platform

3.2.27 property

A property is a valued feature of a class, an interface or a model that can be accessed by two operations, the setter and the getter. A read-only property only has a getter, while a write-only property only has a setter. Properties may be used for controlled access to fields.

See: field

3.2.28 provider [component]

A provider is a component that

- implements an interface (provided interface) whose operations or properties may be invoked by other components (called consumers), or that
- sends event notifications from one of its event sources to other components (called consumers) that have subscribed to this event source with one of their event sinks, or that
- exposes output fields that may be transferred to input fields of other components (called the consumers). The components holding output and input fields are also called source and target.

Contrast: consumer

See: interface, event source, output field, source

3.2.29 publisher [component]

A publisher is a component that

- publishes entry points that may be added to the Scheduler or Event Manager service.

Contrast: consumer

See: entry point, service

3.2.30 reference

A component may reference interfaces of other components via aggregation. It consumes functionality from the other components (called providers) via its references. A reference is typed by an interface, specifying which types of components may be referenced. Furthermore, it may specify the minimum and maximum number of allowed references.

See: aggregation, component, consumer

UML: interface, required interface

3.2.31 run-time [model] instance

A run-time model instance is a run-time instance of a model implementation, and is part of a run-time configuration.

In ANSI/ISO C++, this is usually an in-memory instance (i.e. an object) of a class, created, for example, by the C++ 'new' directive and initialised by a constructor of the class.

Contrast: model implementation

See: [run-time] configuration

3.2.32 [run-time] configuration

A run-time configuration is a run-time network of run-time model instances, which may be created according to the information in an associated assembly, taking the implementation attributes of the model instances into account. The run-time model instances are

- created according to platform bindings in the implementation attribute, and
- configured according to initial values, interface links and event links in the assembly.

Contrast: assembly

See: run-time [model] instance

3.2.33 run-time environment

Synonym: simulation environment, simulator

3.2.34 schedule

A schedule defines how entry points of model instances in an assembly are scheduled. Tasks may be used to group entry points. Typically, a schedule is used together with an assembly in a dynamically configured simulation.

See: dynamically configured simulation

3.2.35 service [interface]

A service is a component that is held as a direct child of the simulation environment, and that provides global functionality to the models. A service is typically queried by name and its functionality is specified by a service interface. As SMP only standardises the interfaces, not their implementation, the terms service and service interface are sometimes used synonymously.

Synonym: simulation service

3.2.36 simulation environment

A simulation environment is a component or application that provides SMP conformant functionality to SMP models. SMP does not specify how a simulation environment is implemented, but rather how models and services interface to the simulation environment. ESA uses a number of different simulation environments, for example EuroSim and SIMSAT.

Synonym: run-time environment

3.2.37 simulator

A simulator is a run-time configuration of a simulation environment and simulation models.

See: simulation environment, model

3.2.38 simulation modelling platform

The simulation modelling platform is the name given to the ECSS-E-TM-40-07 Technical Memorandum which has the objective to enable the effective reuse of simulation models and applications within and between space projects and their stakeholders in order to minimise the overall development costs related to simulation applications.

3.2.39 simulation service

A simulation service is a service used within a simulation.

3.2.40 source [component]

A component that exposes one or more output fields is called a source.

See: output field, provider

3.2.41 statically configured simulation, static configuration

A statically configured simulation is configured from some piece of source code, i.e. model instances are created and connected programmatically. In a statically configured simulation, a change of the model hierarchy, of initial values, of model links, or of model scheduling requires changing some source code.

Contrast: dynamically configured simulation

3.2.42 target [component]

A component that exposes one or more input fields is called a target.

See: consumer, input field

3.3 Abbreviated terms

For the purpose of this Technical Memorandum, the abbreviated terms from ECSS-S-ST-00-01 and the following apply:

Abbreviation	Meaning
AD	applicable document
API	application programming interface
CCM	CORBA component model
COM	component object model
CORBA	common object request broker architecture
COTS	commercial off-the-shelf
DLL	dynamic link library
DSO	dynamic shared object
ECSS	European Cooperation for Space Standardization
ESA	European space agency
GMT	Greenwich mean time
HITL	hardware-in-the-loop
IDL	interface definition language
J2EE	Java 2 enterprise edition
MDA	model driven architecture
MDK	model development kit
MJD	modified julian date
N/A	not applicable
OMG	object management group
OO	object oriented
PIM	platform independent model
PSM	platform specific model
RD	reference document
SMDL	simulation model definition language
SMP	simulation modelling platform
SysML	systems modelling language
TBC	to be confirmed
TBD	to be defined
UML	unified modelling language
UTC	coordinated universal time
UUID	universally unique identifier
XML	extensible mark-up language

4
Simulation modelling platform principles

4.1 Objectives

The main objective of this Technical Memorandum is to enable the effective reuse of simulation models and applications within and between space projects and their stakeholders in order to minimise the overall development costs related to simulation applications. In particular the Technical Memorandum supports model reuse across different simulation environments and exchange between different organisations and missions, improving productivity and medium and long-term reliability in the development of project simulators.

The portability of models between different simulation environments is supported by defining a standard interface between the simulation environment and the models. Models can therefore be plugged into a different simulation environment without requiring any modification to the model source code. This is possible as the interface offered by the simulation environment to the model or by the model to the simulation environment does not change between simulation environments.

The portability of models between different operating systems and hardware is supported by guidelines to the model developer on how to avoid developing models that make use of the operating system, such as making calls to operating specific APIs or hardware specific dependencies.

Further the Technical Memorandum supports the integration of individual models to form a complete system simulation. Improved model integration is closely related to model reuse, as it depends on the availability of well defined reusable models that can be plugged together based on standard mechanisms. The model integration is achieved by a standard format for defining the topology and scheduling of the models involved in a simulator.

Finally the Technical Memorandum supports the improved use of simulation in the system design process, by making it easier to assimilate system-engineering data, or even completely configure a simulation from this data.

4.2 Concepts

The main purpose of SMP is to promote platform independence, interoperability and reuse of simulation models. In order to achieve these goals, two fundamental requirements can be formulated:

- Common Concepts: All SMP models must be built using common high-level concepts addressing fundamental modelling issues. This enables the development of models on an abstract level, which is essential for platform independence and reuse of models.
- Common Type System: All SMP models must be built upon a common type system. This enables different models to have a common understanding of the syntax and semantics of basic types, which is essential for interoperability between different models.

In other words, the first requirement specifies that all models are derived from the same fundamental concepts, and the second requirement specifies that all models be built upon common ground. Thus, specific models 'live' in between these two common layers as is shown in Figure 4-1.

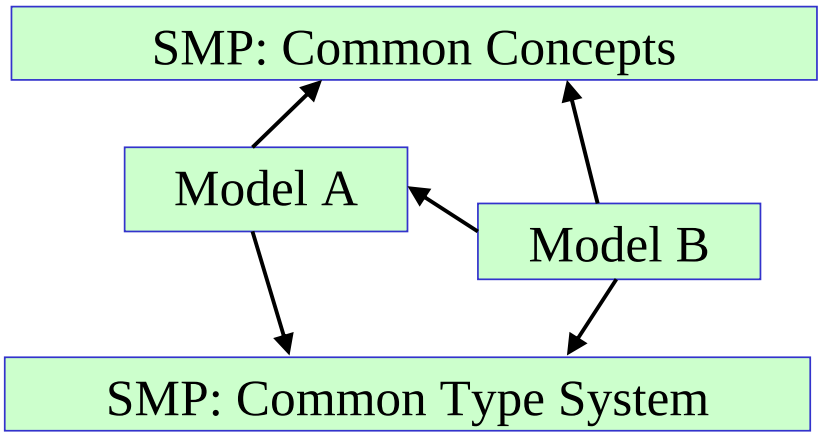

Figure 4-1: Common concepts and type system

The Figure 4-2 illustrates a high-level overview of the main elements of SMP. The figure has three main layers that correspond to different levels of abstraction.

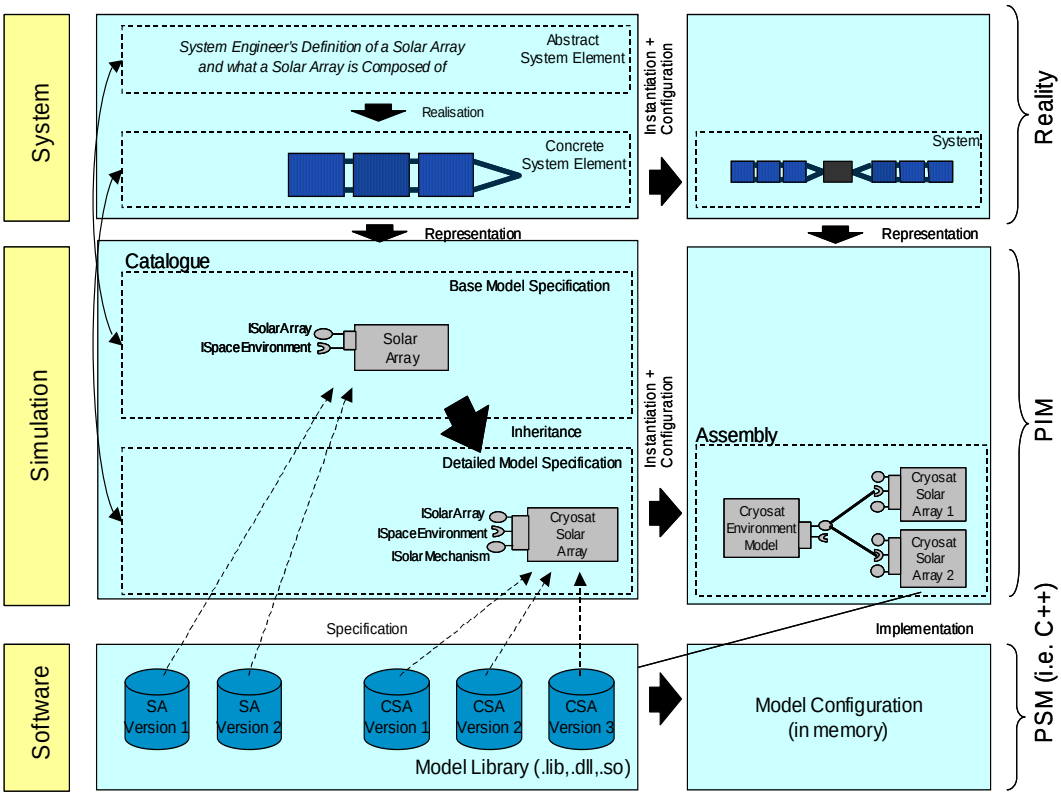

Figure 4-2: SMP high level overview

The top layer represents the real world and deals with the system being modelled. SMP provides no modelling concepts in this layer, although it recognises that there is an important relationship with this layer, particularly with respect to the reuse of system engineering data within a simulation.

The second layer represents the Platform Independent Model (**PIM**) of the simulated system, i.e. the specification level of SMP models. The main concern in this layer is to define model specifications (in a Catalogue file), to define how model instances are integrated and configured (in an Assembly file), and to define how model instances are scheduled (in a Schedule file). The Simulation Model Definition Language (**SMDL**) defines the format of all these XML files; see the Metamodel [Volume 2: Metamodel] for details. Note that the Assembly and Schedule files in this layer are optional and mainly provide support for model integration using external XML files independent of the model implementation. The Catalogue file has added value for every SMP model as it specifies the exact interface of the model and thus may be seen as machine-readable model documentation. This information can be used by code generators to generate model skeleton code for a particular platform mapping, such as ANSI/ISO C++.

The third layer is the Platform Specific Model (**PSM**) of the simulated system, i.e. the implementation level of SMP models. This Technical Memorandum specifies the mapping to ANSI/ISO C++. In the future, further mappings may be defined. Although SMP models may be implemented manually, large parts of the model implementation are concerned with integration and may be generated from information in the Catalogue. To support this, the component model [Volume 3: Component Model] describes the standard interfaces of SMP

components (models and services) and a set of standard simulation services. Finally, the C++ mapping [Volume 4 - C++ Mapping] describes the mapping of metamodel and component model artefacts into ANSI/ISO C++.

The Figure 4-2 also shows a split into two columns. The left hand column corresponds to the definitions of the artefacts, corresponding to types in terms of object oriented (OO) technology. The right hand column corresponds to the artefacts themselves, corresponding to instances in OO technology.

In the example, the modeller first describes in abstract form what a spacecraft solar array is and what it is composed of. This information can be expressed, with respect to the concern of simulation, using a model called Solar Array in an SMP Catalogue. This model may then be used to derive a more specialised solar array models. In the Assembly, instances of these models may be instantiated and configured, resolving references to the interfaces supported by other models. On the implementation level (i.e. ANSI/ISO C++), appropriate model implementations point back to their associated model specification in the Catalogue, allowing tools to provide intelligent user interfaces for the task of selecting a model implementation for a given model specification.

4.3 Architecture

The SMP architecture covers two types of components: A Simulation with Model Instances that provide the application specific behaviour, and a Simulation Environment that provides Simulation Services. This architecture is depicted in the following Figure 4-3.

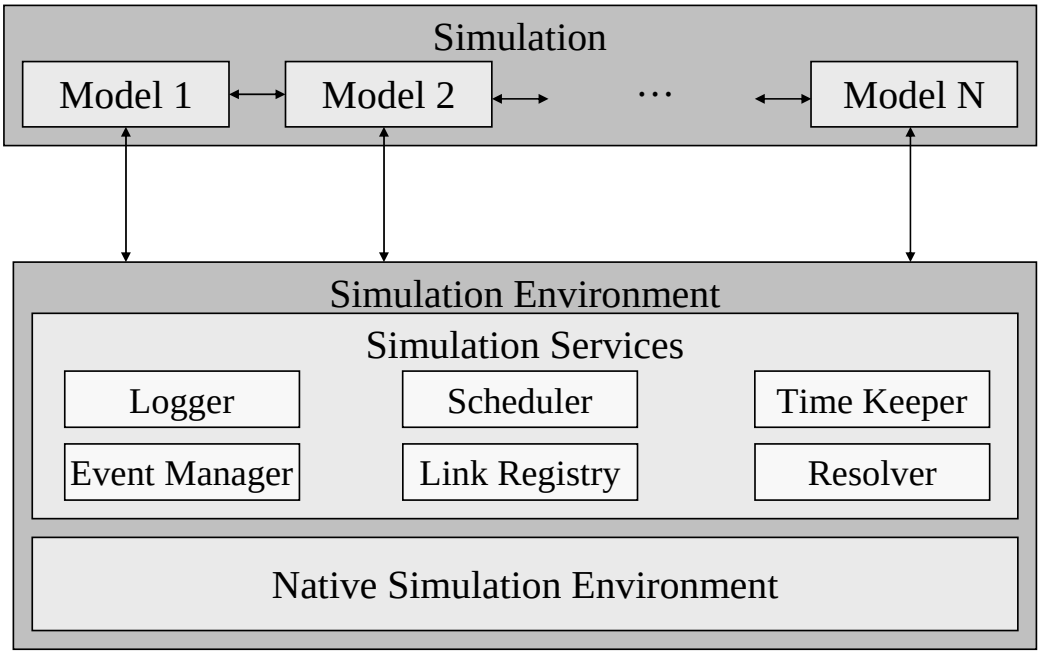

Figure 4-3: SMP architecture

Typically, a simulation environment is based on some existing Native Simulation Environment that it wraps (or adapts) to make it SMP compliant. The simulation environment has to provide six mandatory simulation services:

- Logger: The logger allows logging messages of different kind consistently, for example information, event, warning, and error messages. It is used by services as well as by models.
- Scheduler: The scheduler calls entry points based on timed or cyclic events. It closely depends on the time keeper service.
- Time Keeper: This service provides the four different time kinds of SMP: A relative simulation time, an absolute epoch time, a relative mission time, and Zulu time which relates to the clock time of the computer.
- Event Manager: This service provides mechanisms for global asynchronous events: Event handlers can be registered, events can be broadcasted, and user-specific event types can be defined as well.
- Resolver – This services provides the ability to get a reference to any model within a simulation.
- Link Registry – This services maintains a list of the links between model instances. When a model instance is deleted, the link manager notifies all clients models holding a reference to the deleted model that the reference is no valid.

The arrows in the Figure 4-3 indicate interaction between components. In SMP, communication is typically performed via interfaces . Two different types of interfaces can be identified in this architecture:

- Interfaces between Model and Simulation Environment, and
- Interfaces between Model and Model (inter-model communication).

All these interfaces need to be established before executing models. Therefore, SMP defines different operational phases.

5
Simulator development process

5.1 Introduction

All requirements specified in ECSS-E-ST-40 are applicable to simulation development procurements. The development of simulation software therefore follows the ECSS-E-ST-40 software life-cycle process. De-scoping of requirements can be done by the ECSS-E-ST-40 tailoring process, taking into account the specific simulation software characteristics. The Figure 5-1 shows the ECSS-E-ST-40 software life-cycle process.

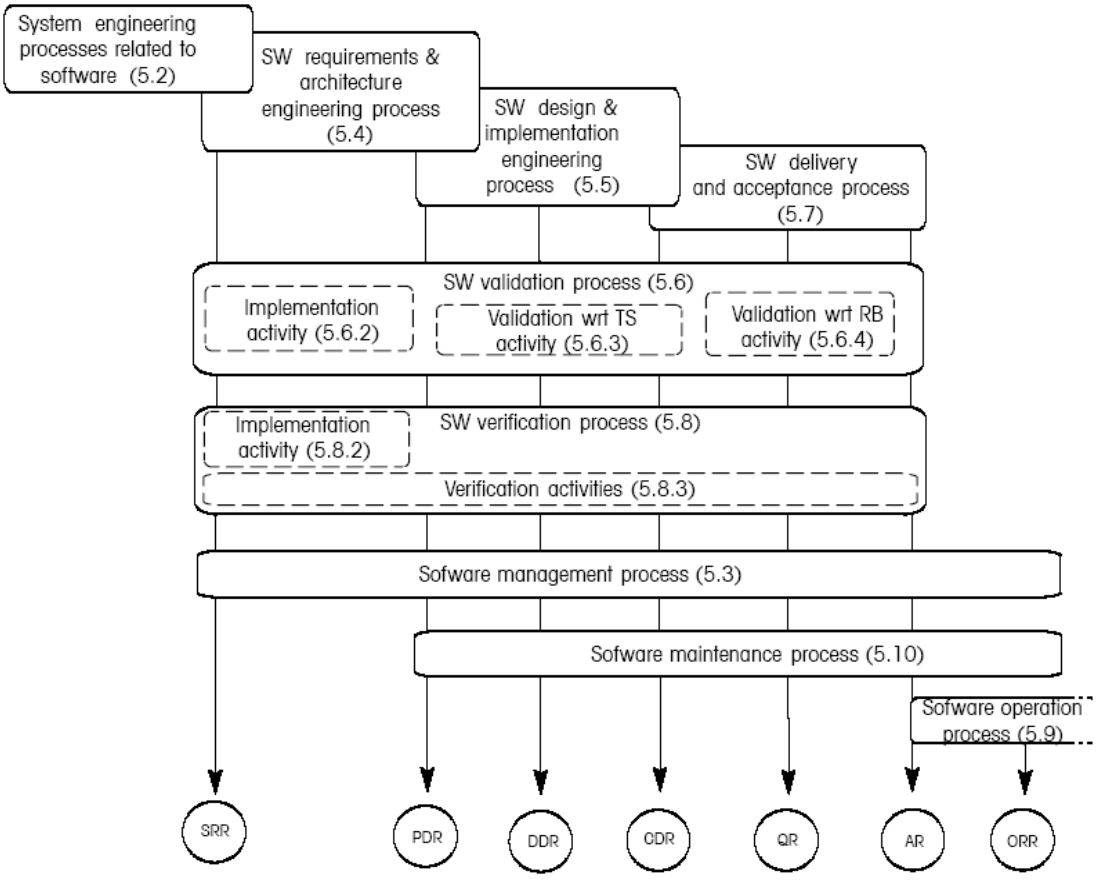

Figure 5-1: ECSS-E-ST-40 software life-cycle process

This Technical Memorandum contains one normative clause (clause 6) which is organized with respect to the software processes and activities breakdown illustrated in Figure 5-1. Each process includes activities which are themselves decomposed into a list of one single or several requirements. The requirements in clause 6 are the special provisions applicable to the development of simulation software.

5.2 Conformance

This Technical Memorandum identifies different levels of conformance that can be applied to a simulation software development. Two levels of conformance are identified:

- Level 1 – Model Exchange (catalogues, packages and configurations): supports the exchange of models between simulation environments.
- Level 2 – Simulator Integration: Supports the integration of simulation models to produce a complete simulation application.

All clauses in this Technical Memorandum are assigned a conformance level in Annex B. All level 1 clauses are applicable to a simulation software development which is defined to be compliant with conformance level 1. All level 1 and level 2 clauses are applicable to a simulation software development which is defined to be compliant with conformance level 2.

The SMP Technical Memorandum differentiates between the different products required in the development of simulation software system to which a conformance level can be assigned. These products are grouped into Conformance Profiles. The following profiles are identified:

- Profile 1 – Development Tools: These are products used to support the development of simulation models and their associated artefacts (see Profile 2).
- Profile 2 – Runtime Infrastructures: These are products used to execute simulation models and their associated artefacts (see Profile 2).
- Profile 3 – Models: These products simulate the behaviour of the system being modelled. Models can be developed with the support of Development Tools (see Profile 1) and are executed within a Run-Time Infrastructure (see Profile 2).

Each of the clauses in this Technical Memorandum is assigned to one or more profiles in Annex B. Typically products in Profile 1 and Profile 2 are COTS that a simulation software development will use or purchase according to the requirements of the project.

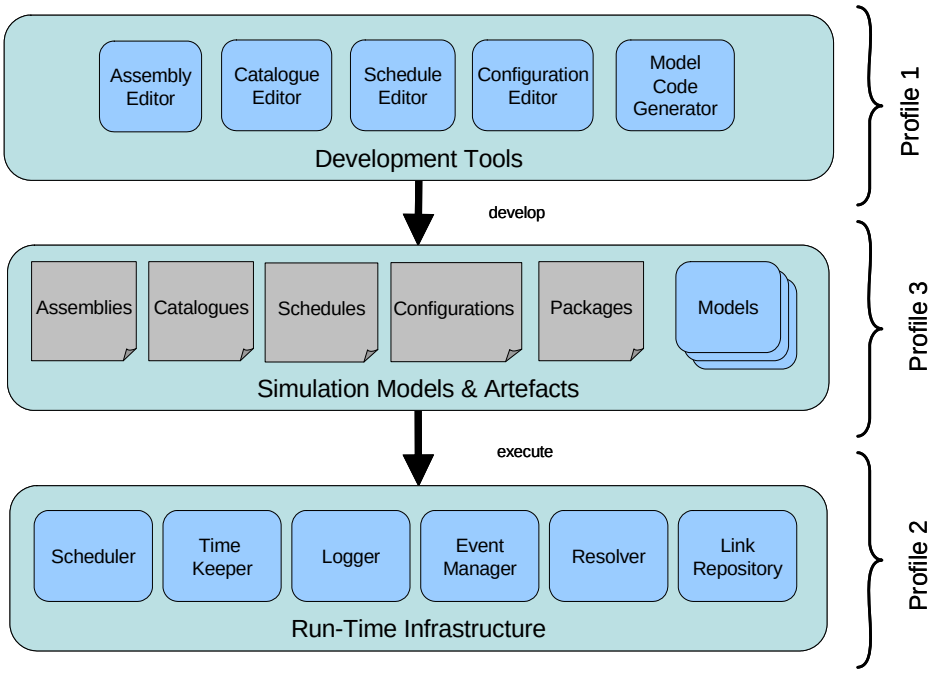

Figure 5-2: SMP conformance profile dependencies

The Figure 5-2 illustrates the dependencies between the conformance profiles. It shows that development tools (Profile 1) are used to develop models (Profile 3) and the run-time infrastructure (Profile 2) executes the models. The conformance profiles applied to a project must be compatible. For example, models developed according to conformance level 1 can be executed by a run-time infrastructure compliant with either level 1 or level 2. However, models developed and integrated according to conformance level 2 can only by executed by a run-time infrastructure also compliant with level 2.

6
Requirements

6.1 Introduction

The requirements outlined in the subsequent clauses are intended to be applicable in simulator development procurements. The requirements have been organised after the process tasks defined in clause 4.

All requirements specified in ECSS-E-ST-40 are applicable to simulation development procurements. This clause includes the additional requirements for the specific provisions applicable to the development of simulation software.

6.2 System engineering process related to software

6.2.1 Simulator infrastructure definition

6.2.1.1 Development tools

6.2.1.1.1 Catalogue editing

a. The simulation infrastructure development tools shall support the editing of SMDL catalogues.

NOTE 1 conformance P1/L1

NOTE 2 see ECSS-E-TM-40-07 Volume 2: Metamodel.

6.2.1.1.2 Catalogue validation

a. The simulation infrastructure development tools shall support the validation of SMDL catalogues against the SMDL validation rules.

NOTE 1: conformance P1/L1

NOTE 2: see ECSS-E-TM-40-07 Volume 2: Metamodel, clause 10

6.2.1.1.3 Package editing

a. The simulation infrastructure development tools shall support the editing of SMDL packages.

NOTE 1: conformance P1/L1

NOTE 2: see ECSS-E-TM-40-07 Volume 2: Metamodel.

6.2.1.1.4 Package Validation

a. The simulation infrastructure development tools shall support the validation of SMDL packages against the SMDL validation rules

NOTE 1: see ECSS-E-TM-40-07 Volume 2: Metamodel, clause 10.

NOTE 2: conformance: P1/L1

6.2.1.1.5 Configuration Editing

a. The simulation infrastructure development tools shall support the editing of SMDL configurations

NOTE 1: see ECSS-E-TM-40-07 Volume 2: Metamodel.

NOTE 2: conformance: P1/L1

6.2.1.1.6 Code Generation

a. The simulation infrastructure development tools shall support the generation of model skeleton code from input SMDL catalogues and packages which is compliant with the C++ Mapping

NOTE 1: see ECSS-E-TM-40-07 Volume 4: C++ Mapping.

NOTE 2: conformance P1/L1

6.2.1.1.7 Assembly Editing

a. The simulation infrastructure development tools shall support the editing of SMDL assemblies

NOTE 1: see ECSS-E-TM-40-07 Volume 2: Metamodel.

NOTE 2: conformance: P1/L2

6.2.1.1.8 Assembly Validation

a. The simulation infrastructure development tools shall support the validation of SMDL assemblies against the SMDL validation rules

NOTE 1: see ECSS-E-TM-40-07 Volume 2: Metamodel, clause 10.

NOTE 2: conformance: P1/L2

6.2.1.1.9 Schedule Editing

a. The simulation infrastructure development tools shall support the editing of SMDL schedules

NOTE 1: see ECSS-E-TM-40-07 Volume 2: Metamodel.

NOTE 2: conformance: P1/L2

6.2.1.1.10 Schedule Validation

a. The simulation infrastructure development tools shall support the validation of SMDL schedules against the SMDL validation rules

NOTE 1: see ECSS-E-TM-40-07 Volume 2: Metamodel, clause 10)

NOTE 2 conformance: P1/L2

6.2.1.2 Runtime Infrastructure

6.2.1.2.1 Model Execution

a. The simulation infrastructure shall support the execution of one or several models compatible with the Component Model (see ECSS-E-TM-40-07 -Volume 3).

NOTE 1 it shall not be necessary to modify model source code or it's resultant binary file to integrate a model into the run-time simulation infrastructure.

NOTE 2 conformance: P2/L1

6.2.1.2.2 Model Initialisation

a. The simulation infrastructure shall allow the initialisation of models from the standard configuration file format

NOTE 1: see ECSS-E-TM-40-07- Volume 2: Metamodel.

NOTE 2 conformance: P2/L1

6.2.1.2.3 Model Loading

a. The simulation infrastructure shall support the loading and initialisation of models defined within a specific assembly

NOTE 1: see ECSS-E-TM-40-07 Volume 2: Metamodel.

NOTE 2: conformance: P2/L2

6.2.1.2.4 Model Scheduling

a. The simulation infrastructure shall support the scheduling of the compliant models by using the schedule definition as specified in this technical memorandum

NOTE 1: see ECSS-E-TM-40-07 Volume 2: Metamodel.

NOTE 2: conformance: P2/L2

6.2.1.2.5 Simulation Services

a. The simulation infrastructure shall support all simulation services specified in the Technical Memorandum

NOTE 1: see ECSS-E-TM-40-07-Volume 3: Component Model.

NOTE 2: conformance: P2/L1

6.2.2 Simulator models requirements identification

6.2.2.1 Software Model Requirements

a. The customer shall identify the high level requirements of the required simulation models.

b. The simulation models requirements shall be derived from an analysis of the specific intended use of the system and of the functionalities and behaviour of the system to be simulated.

c. The following requirements shall be identified in addition:

1. model fidelity level
2. performance requirements
3. identification of algorithms and transfer laws
4. special portability and design implementation requirement

EXPECTED OUTPUT: The following outputs are expected:

a. Simulation models detailed requirements [RB, -; SRR];

b. Prototypes results [RB, -;SRR];

c Conformance:P3/L1

6.2.2.2 Support to Software Models Requirements

a. The customer shall identify and execute analysis and/or prototyping activities in order to establish and assess the high-level simulation requirements.

NOTE the meaning of this requirement is that is the customer responsibility by the RB to define exactly the models to be implemented and to perform any necessary prototyping, breadboard activity in order to ensure the models are the correct ones and that they are ready for software implementation.

EXPECTED OUTPUT: The following outputs are expected:

a. Simulation models detailed requirements [RB, -; SRR];

b. Prototypes results [RB, -;SRR];

c Conformance:P3/L1

6.2.2.3 Simulation Models Validation Planning

a. The simulation model requirements gathering processes shall be aligned to the life cycle of the system to be simulated.

EXPECTED OUTPUT: The following outputs are expected:

a. Software Development plan [MGT, SDP; SRR, PDR];

b Conformance:P3/L1

6.3 Software management process

The requirements on the simulation management process are unchanged from the requirements on the software management process as specified in ECSS-E-40 Part 1B.

a. ECSS-E-ST-40, clause 5.3 shall apply

6.4 Software Requirements and Architecture Engineering Process

6.4.1 Simulator Requirements Analysis

The requirements on the simulation requirements analysis process are unchanged from the requirements on the software requirements analysis process as specified in ECSS-E-40 Part 1B.

a. ECSS-E-ST-40, clause 5.4.2 shall apply

6.4.2 Simulator Architectural Design

6.4.2.1 Architecture Type Breakdown

a. An architectural breakdown (model type breakdown) of the simulator shall be provided and defined in the Catalogue (see Volume 2: Metamodel).:

EXPECTED OUTPUT: The following outputs are expected:

a. Architecture defined in the catalogue [DDF, Catalogue; PDR];

b. Architecture described in Architectural Design Document (SAD) [DDF, SAD;PDR];

c Conformance:P3/L1

6.4.2.2 Architecture Instance Breakdown

a. An architectural breakdown (model instance breakdown) of the simulator shall be provided and defined in the Assembly (see Volume 2: Metamodel).

EXPECTED OUTPUT: The following outputs are expected:

a. Architecture defined in the assembly [DDF, Assembly; PDR];

b. Architecture described in Architectural Design Document (SAD) [DDF, SAD;PDR];

c Conformance:P3/L2

6.4.2.3 Model Scheduling

a. Scheduling Entry Points (see Volume 2: Metamodel) shall be defined for each model that requires scheduling.

EXPECTED OUTPUT: The following outputs are expected:

a. Entry points in model definitions [DDF, Catalogue; PDR];

b. Conformance:P3/L2

6.4.2.4 Simulator Scheduling

a. The scheduling of models shall be defined in a Schedule file (Volume 2: Metamodel).

b. A schedule task or sub-tasks shall be defined whenever it is necessary to guarantee that a set of entry points are to be invoked at the same simulation time (or correlated time) and where it is important to guarantee a predetermined sequence in the execution of the entry points.

EXPECTED OUTPUT: The following outputs are expected:

a. Definition of model scheduling [DDF, Schedule; PDR];

b Conformance:P3/L2

6.4.2.5 Simulator Configurations

a. Simulator configurations supporting different simulation scenarios shall be defined in dedicated assemblies.

EXPECTED OUTPUT: The following outputs are expected:

a. Assemblies supporting different simulation scenario [DDF, Assembly; PDR];

b. Conformance: P3/L2

6.4.2.6 Simulator Configurations Scheduling

a. The scheduling of the models in assemblies supporting different scenarios shall be defined in dedicated Schedule files.

EXPECTED OUTPUT: The following outputs are expected:

a. Assemblies supporting different simulation scenario [DDF, Schedule; PDR];

b. Conformance: P3/L2

6.4.2.7 Model Interface Identification

a. Model interface types shall be identified (name and description) in the Catalogue (see Volume 2: Metamodel).

EXPECTED OUTPUT: The following outputs are expected:

a. Model interface identification [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.8 Model Interface Inheritance

a. Model interface inheritance shall be defined in the Catalogue (see Volume 2: Metamodel).

EXPECTED OUTPUT: The following outputs are expected:

a. Model interface inheritance [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.9 Model Interface Definitions

a. Model interfaces shall be defined (operations, fields, events etc.) in the Catalogue (see Volume 2: Metamodel).

NOTE Only value types, references and other interfaces shall be used as parameters and operations of an interface, i.e. no models shall be used as a property in an interface.

EXPECTED OUTPUT: The following outputs are expected:

a. Model interface definitions [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.10 Model Properties

a. Model properties (operations, properties, value types etc.) shall be defined in the Catalogue (see Volume 2: Metamodel).

EXPECTED OUTPUT: The following outputs are expected:

a. Model interface properties [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.11 Model Type Identification

a. Model types shall be identified (name, description and interfaces) in the Catalogue (see Volume 2: Metamodel).

EXPECTED OUTPUT: The following outputs are expected:

a. Model types [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.12 Model Type Inheritance

a. Model type inheritance shall be defined in the Catalogue (see Volume 2: Metamodel)..

EXPECTED OUTPUT: The following outputs are expected:

a. Model inheritance [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.13 Model Instance Identification

a. Model instances shall be defined in an Assembly (see Volume 2: Metamodel).

NOTE NOTE: All models shall be designed and implemented not to constrain the number of possible instances.

EXPECTED OUTPUT: The following outputs are expected:

a. Model instances [DDF, Assembly; PDR];

b. Conformance:P3/L1

6.4.2.14 Model Instance Dependency Identification

a. Model instance dependencies (interface, field and event links) shall be identified (name and description) in the Assembly (see Volume 2: Metamodel)

EXPECTED OUTPUT: The following outputs are expected:

a. Model instances and their model instances dependencies [DDF,Assembly ; PDR];

b. Conformance:P3/L1

6.4.2.15 Model Platform Access

a. Models performing direct access to services of the underlying platform shall be described in a dedicated document.

EXPECTED OUTPUT: The following outputs are expected:

a. Dedicated platform access models [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.16 Third Party Dependency

a. Any dependencies on third party software libraries shall be clearly defined

EXPECTED OUTPUT: The following outputs are expected:

a. Library dependencies [DDF, SDD; PDR];

b. Conformance:P3/L1

6.4.2.17 Compiler Dependency

a. Any dependencies on specific software compilers shall be clearly defined

EXPECTED OUTPUT: The following outputs are expected:

a. Compiler dependencies [DDF, SDD; PDR];

b. Conformance:P3/L1

6.4.2.18 Model Persistence

a. Persistence of models may be done through the IPersist interface

NOTE It is the responsibility of the model to define what it wants to persist, this can be internal data, internal fields.

EXPECTED OUTPUT: The following outputs are expected:

a. Models realising the IPersist interface [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.19 Simulator Infrastructure Access

a. Models performing direct access to non-standardised SMP simulation services shall be described in a dedicated document.

EXPECTED OUTPUT: The following outputs are expected:

a. Infrastructure dependencies [DDF, SDD; PDR];

b. Conformance:P3/L1

6.4.2.20 Simulation Infrastructure Dependencies

a. All model dependencies on simulation services shall be defined and explicitly reference the services defined in the Service Catalogue (see Volume 2: Metamodel)

EXPECTED OUTPUT: The following outputs are expected:

a. Model simulation infrastructure dependencies [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.21 Simulation Data Types

a. All simulation data types used shall be defined in Catalogue(s) and shall be derived from the standardised types and primitive elements.

EXPECTED OUTPUT: The following outputs are expected:

a. Catalogue(s) defining data types [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.22 Catalogue Element Descriptions

a. All Catalogue elements shall contain an associated description

EXPECTED OUTPUT: The following outputs are expected:

a. Element descriptions [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.23 Assembly Element Descriptions

a. All Assembly elements shall contain an associated description

EXPECTED OUTPUT: The following outputs are expected:

a. Element descriptions [DDF, Assembly; PDR];

b. Conformance:P3/L1

6.4.2.24 Model Meta Data

a. All models shall contain associated metadata describing the model characteristics.

NOTE This can be captured directly in the Catalogues using attributes or using references to external data (e.g. data sheet of a modelled component)

EXPECTED OUTPUT: The following outputs are expected:

a. Model interface realisations (simulation services, events, interfaces and fields) [DDF, Catalogue; PDR];

b. Conformance: P3/L1

6.4.2.25 Model Communication

a. All inter-model communication shall be achieved by means of the mechanisms available in the standardised Component Model (see Volume 3: Component Model)

NOTE The goal of this requirement is to encapsulate the models to achieve reusable models.

EXPECTED OUTPUT: The following outputs are expected:

a. Model interface realisations (simulation services, events, interfaces and fields) [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.26 Model Aggregation

a. Model aggregation shall be used whenever a model contains references to other models but does not command the lifecycle of the other models.

NOTE 1 This is the case when a model needs to access an interface provided by another model

NOTE 2 A usage example of this approach is when a model consumes an interface, e.g. a power distribution model which controls all power consuming models.

EXPECTED OUTPUT: The following outputs are expected:

a. Model aggregation [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.27 Model Composition and Containment

a. Model composition shall be used when a model owns a set of models, i.e. the life cycle of the dependent models is controlled by the parent model.

b. Model containment shall be used to group models of different categories

EXPECTED OUTPUT: The following outputs are expected:

a. Model composition and containment [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.4.2.28 Managed Models

a. All simulation models shall be managed models and implement the IManagedModel interface (see Volume 2: Metamodel)

NOTE This interface allows for a simulation infrastructure to dynamically load and create model instances from an assembly file. It also allows for the dynamic scheduling of the model Entry Points. The support of this feature does not imply additional effort as it can be derived from the MDK helper classes.

EXPECTED OUTPUT: The following outputs are expected:

a. Model realising the IManagedModel interface [DDF, Catalogue; PDR];

b. Conformance :P3/L1

6.4.2.29 Dynamic Invocation

a. All simulation models shall allow for dynamic invocation of their operations.

b. This shall be achieved by implementing the IDynamicInvocation interface of the Component Model (see Volume 3: Component Model)

EXPECTED OUTPUT: The following outputs are expected:

a. Models realising the IDynamicInvocation interface [DDF, Catalogue; PDR];

b. Conformance:P3/L1

6.5 Software Design and Implementation Engineering Process

6.5.1 Introduction

6.5.1.1 Model Property Initial Values

a. Initial values of model instance properties and fields shall be defined in an Assembly (see Volume 2: Metamodel).

EXPECTED OUTPUT: The following outputs are expected:

a. Model initial values [DDF, Assembly; CDR];

b. Conformance :P3/L2

6.5.1.2 Model Integration

a. All model integration shall be achieved by means of the definition present in assembly and schedule files that are used to instantiate, initialise and schedule the simulator.

EXPECTED OUTPUT: The following outputs are expected:

a. Simulator definition [DDF, Assembly; CDR];

b. Definition of simulator scheduling [DDF, Schedule; CDR]

b. Conformance :P3/L2

6.5.1.3 Model Dependency Definition

a. Model dependencies (interface, field and event links) shall be defined including the consumer model and the provider model in the Assembly (see Volume 2: Metamodel).

EXPECTED OUTPUT: The following outputs are expected:

a. Model dependencies [DDF, Assembly; CDR];

b. Conformance :P3/L2

6.5.1.4 Circular Dependencies

a. There shall be no circular dependency between catalogue files.

EXPECTED OUTPUT: The following outputs are expected:

a. Model dependencies [DDF, Catalogue; CDR];

b. Conformance :P3/L1

6.5.1.5 Binary Compatibility

a. All models shall be built in accordance with the Binary Constraints Specification for the selected platform mapping (see Volume 4: C++ Mapping – clause 6.6.3).

EXPECTED OUTPUT: The following outputs are expected:

a. Binary compatible models [DDF, Binaries; CDR];

b. Conformance :P3/L1

6.5.1.6 Model Skeleton Generation

a. It shall be possible to generate all simulator source code skeletons from the information provided by the catalogues using a code generator compatible with the C++ mapping (see Volume 4: C++ Mapping).

EXPECTED OUTPUT: The following outputs are expected:

a. Model skeleton source code [DDF, Code; CDR];

b. Conformance :P3/L1

6.5.1.7 Model Packaging

a. The packaging of any elements into binary files (Dynamic Linked Libraries, Shared Objects etc.) shall be defined in a Package file (see Volume 4: C++ Metamodel).

b. Dependencies between packages shall be defined by means of the dependency relationship (enabling the establishment of a package dependency tree).

EXPECTED OUTPUT: The following outputs are expected:

a. Model binary packaging [DD , Package; CDR];

b. Conformance :P3/L2

6.6 Software Validation Process

6.6.1 Introduction

The simulation model validation process comprises the definition of model validation tasks as well as the execution of these tasks. This process is consistent with the software engineering process. Synchronization points are identified through the milestones. It is this process, in particular which interacts with and complements the activities and tasks identified in the System Engineering process related to Software.

6.6.1.1 Simulation Models Validation Requirements

a. The customer shall identify any special requirements, constraints, inputs data source as inputs to the model validation program implementation.

EXPECTED OUTPUT: The following outputs are expected:

a. Simulation models validation requirements [RB, -; SRR];

b. Conformance :P3/L1

6.6.1.2 Simulation Models Validation Planning

a. The supplier shall identify, establish and document a Simulation Modelling Validation program for the simulation models in order to ensure the simulation models correctly represent the behaviour of the simulated items, at the required fidelity level.

NOTE Depending on the simulated items, the validation program can utilize hardware tests results or correlate simulator outputs wrt. other system results.

EXPECTED OUTPUT: The following outputs are expected:

a. Simulation model validation plan [DJF,- ; CDR];

b. Conformance :P3/L1

6.6.1.3 Simulation Validation Program Phasing

a. The supplier shall plan, phase and execute the simulation validation activities in strict synchronisation with the simulation software validation activities, in order to ensure the resulting software item is verified and validated and tested.

EXPECTED OUTPUT: The following outputs are expected:

a. Simulation model validation testing specification [DJF, -; CDR];

b. System Validation Testing Specifications [DJF,-; CDR]

c. Conformance :P3/L1

6.6.1.4 Simulation Validation Testing Preparation

a. The supplier shall identify and document the detailed model validation activities.

EXPECTED OUTPUT: The following outputs are expected:

a. Simulation model validation testing specification [DJF, -; CDR];

b. System validation testing specifications (at simulator level) [DJF,-; CDR]

c. Conformance :P3/L1

6.6.1.5 Simulation Validation Testing Execution

a. The supplier shall execute and document the simulation validation activities.

EXPECTED OUTPUT: The following outputs are expected:

a. Simulation model validation testing reports [DJF, -; CDR];

b. System validation testing reports (at simulator level) [DJF,-; CDR]

c. Conformance:P3/L1

6.6.1.6 Interface Tests

a. Tests shall be defined for each defined model interface.

b. The code coverage of the tests shall be defined on an interface basis.

EXPECTED OUTPUT: The following outputs are expected:

a. Interface tests [DJF, Test specifications/plans; CDR];

b. Conformance: P3/L1

6.6.1.7 Model Tests

a. Each model shall be tested in isolation (stubs shall be provided for the required interfaces) to validate the correct realisation of the provided interfaces.

EXPECTED OUTPUT: The following outputs are expected:

a. Model tests [DJF, Test specifications/plans; CDR];

b. Conformance :P3/L1

6.7 Software Delivery and Acceptance Process

a. ECSS-E-ST-40, clause 5.7 shall apply

6.8 Software Verification Process

a. ECSS-E-ST-40, clause 5.8 shall apply

6.9 Software Operation Process

a. ECSS-E-ST-40, clause 5.9 shall apply

6.10 Software Maintenance Process

6.10.1 Modification Implementation

a. The consistency between the Catalogue and the model implementation code shall be maintained throughout the lifecycle of a model.

EXPECTED OUTPUT: The following outputs are expected:

a. Maintained catalogues [DDF,Catalogues; ; PDR];

b. Conformance :P3/L1

Annex A (normative) Simulator artefacts

Table A-1 describes the simulator software artefacts which are delivered by the simulator software supplier for the different reviews defined within the ECSS-E-ST-40 software life-cycle process.

Table A-1: Simulator artefacts

Artefact	Description	Review
Catalogue	A catalogue contains namespaces as a primary ordering mechanism, where each namespace may contain types. These types can be language types, which include model types as well as other types like structures, classes, and interfaces. Additional types that can be defined are event types for inter-model events, and attribute types for typed metadata. The language for describing entities in the catalogue is the SMP Metamodel, or SMDL.	PDR,CDR,QR,AR
Configuration	A configuration supports the specification of arbitrary field values on component instances in the simulation hierarchy. This can be used to initialise or reinitialise the simulation.	PDR,CDR,QR,AR
Assembly	An assembly describes the configuration of a set of model instances, specifying initial values for all fields, actual child instances, and type-based bindings of interface, events, and data fields. An assembly may be independent of any model implementations that fulfil the specified behaviour of the involved model types, i.e. an assembly may specify a scenario of model types without specifying their implementation. On the other hand, each model instance in an assembly is bound to a specific model in the catalogue for its specification.	PDR,CDR,QR,AR
Schedule	A schedule defines how entry points of model instances in an assembly are scheduled. Tasks may be used to group entry points. Typically, a schedule is used together with an assembly in a dynamically configured simulation	PDR,CDR,QR,AR
Package	A package holds an arbitrary number of implementation elements. Each of these implementations references a type in a catalogue that	CDR,QR,AR

Artefact	Description	Review
	shall be implemented in the package. In addition, a package may reference other packages as a dependency.	
Model Implementation	A model implementation implements the behaviour of the model. The model implementation may be delivered by the simulation model supplier at either source code or binary code level. Source Code – the model supplier provides the simulator integrator with the complete source code of the model which is used to build the binary version of the model. Binary Code - the model supplier provides the simulator integrator with model code in binary format with the associated model header files. In this case, the supplier must follow the constraints for enabling the exchange of binary model exchange, as described in Volume 4: C++ Mapping. This may be done when the source code cannot be provided because of IPR issues.	CDR,QR,AR

Annex B (normative) Conformance

Table B-1 presents the conformance level and applicable conformance profile(s) of the clauses contained in clause 6. The conformance levels and profiles are defined in clause 5.2.

Table B-1: Clause conformance

Clause	Page	Conformance Profile	Conformance Level
6.2.1.1.1a	28	P1	L1
6.2.1.1.2a	28	P1	L1
6.2.1.1.3a	28	P1	L1
6.2.1.1.4a	29	P1	L1
6.2.1.1.5a	29	P1	L1
6.2.1.1.6a	29	P1	L1
6.2.1.1.7a	29	P1	L2
6.2.1.1.8a	29	P1	L2
6.2.1.1.9a	29	P1	L2
6.2.1.1.10a	29	P1	L2
6.2.1.2.1a	30	P2	L1
6.2.1.2.2a	30	P2	L1
6.2.1.2.3a	30	P2	L2
6.2.1.2.4a	30	P2	L2
6.2.1.2.5a	30	P2	L1
6.2.2.1a	30	P3	L1
6.2.2.2a	31	P3	L1
6.2.2.3a	31	P3	L1
6.4.2.1a	32	P3	L1
6.4.2.2a	32	P3	L2
6.4.2.3a	33	P3	L2
6.4.2.4a	33	P3	L2

Clause	Page	Conformance Profile	Conformance Level
6.4.2.5a	33	P3	L2
6.4.2.6a	33	P3	L2
6.4.2.7a	34	P3	L1
6.4.2.8a	34	P3	L1
6.4.2.9a	34	P3	L1
6.4.2.10a	34	P3	L1
6.4.2.11a	34	P3	L1
6.4.2.12a	35	P3	L1
6.4.2.13a	35	P3	L2
6.4.2.14a	35	P3	L2
6.4.2.15a	35	P3	L1
6.4.2.16a	35	P3	L1
6.4.2.17a	36	P3	L1
6.4.2.18a	36	P3	L1
6.4.2.19a	36	P3	L1
6.4.2.20a	36	P3	L1
6.4.2.21a	36	P3	L1
6.4.2.22a	37	P3	L1
6.4.2.23a	37	P3	L2
6.4.2.24a	37	P3	L1
6.4.2.25a	37	P3	L1
6.4.2.26a	37	P3	L1
6.4.2.27a	38	P3	L1
6.4.2.28a	38	P3	L1
6.4.2.29	38	P3	L1
6.5.1.1a	39	P3	L2
6.5.1.2a	39	P3	L2
6.5.1.3a	39	P3	L2
6.5.1.4a	39	P3	L1
6.5.1.5a	40	P3	L1
6.5.1.6a	40	P3	L1
6.5.1.7	40	P3	L1
6.6.1.1a	41	P3	L1

Clause	Page	Conformance Profile	Conformance Level
6.6.1.2a	41	P3	L1
6.6.1.3a	41	P3	L1
6.6.1.4a	41	P3	L1
6.6.1.5a	42	P3	L1
6.6.1.6	42	P3	L1
6.6.1.7a	42	P3	L1
6.10.1	43	P3	L1

Bibliography

ECSS-S-ST-00 ECSS system – Description, implementation and general requirements