ECSS-E-ST-40-08C
5 August 2025

Space engineering

Simulation modelling platform - Level 2

ECSS Secretariat
ESA-ESTEC
Standards Section
Noordwijk, The Netherlands

Foreword

ECSS is a cooperative effort of the European Space Agency, national space agencies and European industry associations for the purpose of developing and maintaining common standards. Requirements in this Standard are defined in terms of what shall be accomplished, rather than in terms of how to organize and perform the necessary work. This allows existing organizational structures and methods to be applied where they are effective, and for the structures and methods to evolve as necessary without rewriting the standards.

This Standard has been prepared by the ECSS-E-ST-40-08C Working Group, reviewed by the ECSS Executive Secretariat and approved by the ECSS Technical Authority.

Disclaimer

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

Change log

ECSS-E-ST-40-08C 5 August 2025	First issue

Table of contents

Figures

Introduction

Space programmes have developed simulation software for a number of years, 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 software are often developed. This standard addresses the issues related to portability and reuse of simulators. For the simulator specifications, it is based on the Simulation Modelling Platform Level 1 standard ECSS-E-ST-40-07 and it reuses the concepts developed in the work performed by ESA in the development of the Simulator Model Portability framework SMP2 starting from the mid-end of the nineties.

This standard integrates the ECSS-E-ST-40 with additional requirements which are specific to the development of simulation software. The formulation of this standard takes into account:

- The existing ISO 9000 family of documents, and
- The Simulation Model Portability specification version 1.2.
- The Simulation Modelling Platform standard ECSS-E-ST-40-07.

The intended readership of this standard is the simulator software customer and supplier.

1 Scope

ECSS-E-ST-40-08 is a standard based on ECSS-E-ST-40 for the engineering of artefacts used to instantiate and to configure simulators.

ECSS-E-ST-40-08 complements ECSS-E-ST-40 and ECSS-E-ST-40-07 in being more specific to simulator artefacts for integration. Simulator artefacts include assembly, schedule and link base files. The standard enables the effective reuse of these artefacts within and between space projects and their stakeholders. In particular, the standard supports assemblies, schedules and link bases reuse across different simulation environments and exchange between different organizations and missions.

ECSS-E-ST-40-08 allows:

- to instantiate and to configure elements defined in ECSS-E-ST-40-07, which are namely the simulation model (including thus its sub-components and related configuration values and interfaces), the simulation events and their scheduling.
- to define connections or links between simulation models.
- to define a standard application process of the artefacts at start-up of a simulator compliant to ECSS-E-ST-40-07.

This standard can be used as an additional standard to ECSS-E-ST-40 and ECSS-E-ST-40-07 providing the additional requirements which are specific to simulator software integration.

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

Applicability

This standard lays down requirements for simulator artefacts including assembly, schedule and link base files. The requirements cover the definitions of these artefacts for the purpose of re-use and exchange to allow a simulator to be deployed and run in any conformant simulation environment.

A consequence of being compliant to this standard for a simulator artefact is the possibility of being reused in several simulation facilities or even in several projects. However, adherence to this standard does not imply or guarantees simulator artefact reusability, it is only a precondition. Other characteristics of the simulator artefacts, to be defined outside this standard, such as the functional content of the artefacts, the simulated spacecraft equipment, the simulator structure, information and data dependent on the space mission, etc. are also heavily affecting the potential for an artefact to be reused. In addition, agreements need to be reached on simulator validation status as well as legal issues and export control restrictions.

Therefore, this standard enables but does not mandate, impose nor guarantee successful simulator artefact re-use and exchange.

Simulator artefact reuse in this standard is meant at the semantics and format level of the content of these artefacts.

2
Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this ECSS Standard. 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 general requirements
ECSS-E-ST-40-07	Simulation modelling platform
[SMPL1_FILES]	ECSS_SMP_L1_(31March2025).zip – SMP Level 1 C++ Header files and XML schemas (Available from ECSS website).
[SMPL2_FILES]	ECSS_SMP_L2_(31March2025).zip – SMP Level 2 C++ Header files and XML schemas (Available from ECSS website).

3
Terms, definitions and abbreviated terms

3.1 Terms from other standards

a. For the purpose of this Standard, the terms and definitions from ECSS-S-ST-00-01 and ECSS-E-ST-40 apply.

b. For the purpose of this Standard, the terms and definitions from ECSS-E-ST-70 apply, in particular for the following term:

   1. mission

c. For the purpose of this Standard, the terms and definitions from ECSS-E-ST-40-07 apply.

3.2 Terms specific to the present standard

3.2.1 assembly

simulator artefact that contains a hierarchy of model instances and assembly instances

3.2.2 assembly instance

occurrence of an external assembly created inside a given assembly

3.2.3 child model instance

model instance contained in a parent model instance via the composition relationship

3.2.4 component configuration

element contained in an assembly or in an assembly instance, that allows configuring one model instance.

3.2.5 dataflow link

synonym of field link

3.2.6 event link

link connecting an event source to an event sink

NOTE Consumed data is the event type and the corresponding data

3.2.7 field link

link connecting an output field to an input field

NOTE Consumed data is the field value

3.2.8 field automatic-propagation

synonym of field self-propagation

3.2.9 field propagation

copy of the value of the output field to all connected input fields

3.2.10 field scheduled-propagation

field propagation is performed by the simulation environment

NOTE This propagation way is scheduled in a simulation event. Therefore, this is called a scheduled propagation

3.2.11 field self-propagation

field propagation is performed by the output field, which can propagate the value immediately upon value change.

NOTE No scheduling is necessary for this kind of field propagation

3.2.12 field value

model field configuration value contained in an assembly or in a component configuration

NOTE Value can be simple when the field is typed by a primitive type such as Bool, Char8, Int32, Float64, Duration, DateTime, String8… or by a simple type. Value can also be enumeration, array, string and structure value. See ECSS-E-ST-40-07 for details

3.2.13 global event

event representing a notification broadcast by the Event Manager service to all components participating to the simulator

NOTE Global events are not to be confused with simulation events, which are managed by the Scheduler service

3.2.14 global event subscription

configuration element contained in an assembly or in a component configuration that allows subscribing an entry point to a global event

3.2.15 interface link

link connecting a reference to the model instance that provides, or implements, a given interface

NOTE Consumed data is the operations of the interface

3.2.16 link

connection between two resolvable objects to allow exchange of data between them

NOTE As resolvable objects are either model instances or child objects of model instances, a link can be considered as a connection between two model instances

3.2.17 link base

simulator artefact that contains a collection of links

3.2.18 link client path

in a link, path to the resolvable object that receives the information

NOTE For the interface link, it is the path of the model instance that implements the interface operations. For the event link, it is the event sink path. For the field link, it is the input field path

3.2.19 link client model instance

model instance that contains the resolvable object identified by a link client path

3.2.20 link owner path

in a link, path to the resolvable object that uses the link to deliver information

NOTE For the interface link, it is the path of the model instance that owns the reference. For the event link, it is the event source path. For the field link, it is the output field path

3.2.21 link owner model instance

model instance that contains the resolvable object identified by a link owner path

3.2.22 model fully qualified type name

concatenation of the model namespaces and the model name separated by *"::"*

NOTE The fully qualified type name is a C++ concept that applies to C++ classes. As a model is mapped on a C++ class, the model fully qualified type name has the same meaning as in C++

3.2.23 model instance connection

synonym of link

3.2.24 model integrator

in a simulator artefact exchange relationship, role of the user who performs integration of simulator artefacts in a simulator

3.2.25 model provider

in a simulator artefact exchange relationship, role of the user who develops and provides simulator artefacts

NOTE Thus, a model provider delivers simulator artefacts to the model integrator, who is in charge of integrating these in a simulator

3.2.26 operation call

configuration element contained in an assembly or in a component configuration, that allows calling a model operation by dynamic invocation

3.2.27 parent model instance

model instance that contains model instances via the composition relationship

3.2.28 property value

model property configuration value contained in an assembly or in a component configuration

NOTE Property values can be typed like field values.

3.2.29 resolvable object

object implementing *Smp::IObject* (see ECSS-E-ST-40-07), found in the simulator hierarchy and identified by a path, which can be resolved by the Resolver service

NOTE Resolvable objects that can appear in the simulator artefacts include model instance, field, property, event source, event sink, entry point and operation

3.2.30 resolvable object path

character string obtained by the concatenation of the names of the components or object instances on the path separated by the separator "/" with the terminal object name

NOTE A path is unique in an assembly and is unique in a simulator. The path is thus an unambiguous identifier for an object in the simulator

3.2.31 sub-model instance

synonym of child model instance

3.2.32 simulator artefact

general term used for an XML metadata file corresponding to the assembly, the schedule, the link base or the SMP Level 1 configuration

NOTE SMP Level 1 configuration is defined in ECSS-E-ST-40-07

3.2.33 simulator artefact loading

process by which the simulator artefact is read and stored into the computer memory

3.2.34 simulator artefact application

process by which the previously loaded artefact in the computer memory is actually instantiated in the simulator that is being configured

NOTE Depending on the implementation optimization, application can occur immediately after loading so that it is not necessary to store the elements in the computer memory

3.2.35 simulator artefact instantiation

synonym of simulator artefact application

3.2.36 sub-assembly

synonym of assembly instance

3.2.37 template argument

element that allows to define parameters by substitution for a simulator artefact

NOTE Template arguments are pre-processed when the artefact is loaded before the actual application of the artefact in the simulator

3.3 Abbreviated terms

For the purpose of this Standard, the same abbreviated terms and symbols from ECSS-S-ST-00-01, ECSS-E-ST-40-07 and the following apply:

Abbreviation	Meaning
XSD	XML Schema Definition

3.4 Conventions

The following conventions apply throughout this document:

a. In the list of terms provided in 3.2, an underlined term indicates that there exists a definition of that term as well in the same clause.

b. The main purpose of this Standard is to specify the requirements for the SMP Level 2 information model, which is expressed with the XSD provided in the Annexes. When the descriptive text refers to an Element or a Concept defined in the information model, the corresponding name in the information model is used.

c. For the specification of software interfaces, conventions in ECSS-E-ST-40-07 are followed.

d. The present and past tenses are used in this Standard to express statements of fact, and therefore they imply descriptive text.

e. The term "SMP Object" is used to refer to a software component that implements *Smp::IObject* as specified in ECSS-E-ST-40-07.

f. The word "feature" has the same meaning as in ECSS-E-40-07. It means an "attribute", a "field", an UML-like "property" of an element. As all these have also their own meaning in SMP Level 1, "feature" is used instead. A "feature" map at XML level to an "XML attribute" or a "child XML element". When referring a "feature", the name given in the information model is used in the descriptive text.

g. The word "collection" is used for an element in the information model that has multiplicity 0..*. Case 0 corresponds to an empty collection, or the absence of the corresponding XML element. The use of "may contain" is a synonym of "contains a collection of".

h. When used in a requirement, "parent model instance" refers to the model instance that owns the element, which is the focus subject of the current requirement.

3.5 Nomenclature

The following nomenclature applies throughout this document:

a. The word "shall" is used in this Standard to express requirements. All the requirements are expressed with the word "shall".

b. The word "should" is used in this Standard to express recommendations. All the recommendations are expressed with the word "should".

NOTE It is expected that, during tailoring, recommendations in this document are either converted into requirements or tailored out.

c. The words "may" and "need not" are used in this Standard to express positive and negative permissions, respectively. All the positive permissions are expressed with the word "may". All the negative permissions are expressed with the words "need not".

d. The word "can" is used in this Standard to express capabilities or possibilities, and therefore, if not accompanied by one of the previous words, it implies descriptive text.

NOTE In ECSS "may" and "can" have completely different meanings: "may" is normative (permission), and "can" is descriptive.

4 Principles

4.1 Objectives

As the objective of this standard aims at specifying the SMP Level 2 information model supporting the definition of SMP Level 2 artefacts, the principles are defined in conceptual and functional terms, which are normally independent of the chosen implementation software technology or exchange format such as XML.

This clause provides definitions and explanations for the following SMP Level 2 elements and concepts:

a. Assembly

b. Model Instance: (Root)Model Instance and (Sub-)Model Instance

c. Assembly Instance

d. Link

e. Template Argument

f. Component Configuration

g. Field Value

h. Property Value

i. Operation Call

j. Global Event Subscription

k. Schedule

l. Epoch Time and Mission Start

m. Task

n. Activity

o. Event

Explanations of the concepts follow a top-down approach starting from the Assembly, Schedule and Link Base architectures. In the text, an element is referred to with the name like it appears in the above list.

The used exchange format, which is XML for ECSS-E-ST-40-08, is then briefly explained in Clause 4.3.

4.2 Architecture

4.2.1 General

The SMP Level 2 simulator architecture is built on the basis of the Assembly, Schedule and Link Base artefacts. The architectures of these and their building blocks are explained in this clause.

4.2.2 Assembly Architecture

4.2.2.1 General

The Assembly is the main building block of a simulator. In order to build-up a simulator, at least one Assembly needs to be provided.

The Assembly defines a hierarchy of model instances, their inter-connections and the elements needed for their configurations. Following elements are the building blocks of a simulator assembly:

a. Exactly one Model Instance, which is the top-level or root model instance of the hierarchy of model instances defined in this Assembly.

b. Zero or more Template Arguments, which define optional parameters for controlling at instantiation time user variables defined in the Assembly.

c. Zero or more Component Configurations, each defining the configuration for one Component, that is a model instance node, in the Assembly hierarchy.

The root Model Instance can contain child Model Instances (i.e. Sub-Model Instances) or child Assembly Instances as well as Links (i.e. model connections) and corresponding elements that allow defining their configurations (i.e. Field Value, Property Value, Operation Call or Global Event Subscription). Each Sub-Model Instance has the same structure as its parent instance, that is, it can contain the same kinds of elements as its parent.

Figure 4-1 shows the overview of an Assembly with all possible elements which can be contained therein:

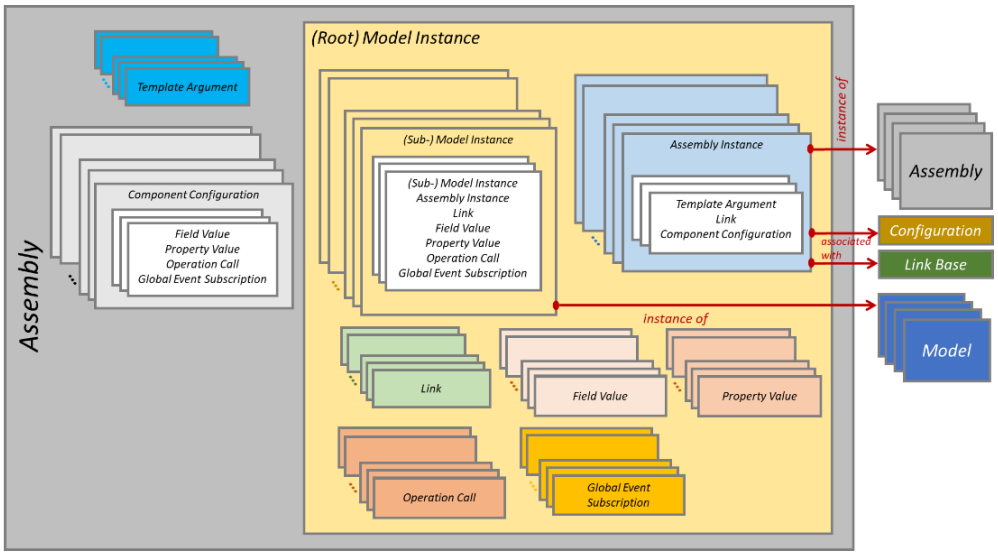

Figure 4-1: Assembly Architecture

In Figure 4-1, (Root) Model Instance and (Sub-)Model Instance are both model instances with a subtle difference:

- (Root) denotes the root node of the Assembly, which is not contained in another model instance.
- (Sub-) denotes that the node is well contained in another model instance.

The arrow "instance of" shown in Figure 4-1 means that a Model Instance (respectively Assembly Instance) needs to reference the Model (respectively Assembly) from which it has been instantiated. Note that Models are elements defined in ECSS-E-ST-40-07.

The arrow "associated with" shown in Figure 4-1 means that an Assembly Instance can be associated with a Configuration file, which contains additional configuration information and/or a Link Base file, which contains additional link information, for that Assembly Instance.

4.2.2.2 Model Instance

Each model instance in the Assembly is identified by a unique Path relative to the top-level or root instance.

Each Model Instance, referred to as the current Model Instance below, contains:

a. A collection of child Model Instances.

b. A collection of child Assembly Instances, which are instantiated from Assemblies and not from Models.

c. A collection of Links (or Model Instance Connections) where a Link can be an Interface Link or an Event Link or a Field Link: these Links connect only Model Instances that are direct or indirect children of the current Model Instance, or the current Model Instance itself.

d. A collection of Field Values whose purpose is to configure fields of the current Model Instance.

e. A collection of Property Values whose purpose is to configure properties of the current Model Instance.

f. A collection of Operation Calls whose purpose is to call operations for the configuration of the current Model Instance.

g. A collection of Global Event Subscription whose purpose is to register Entry Points of the current Model Instance with global events.

4.2.2.3 Assembly Instance

Next to the Model Instances, the Assembly can contain Assembly Instances created from external Assemblies, these external Assemblies following as well the architecture shown in Figure 4-1. Identical Assemblies, which define the same hierarchy and model connections but the related configuration values can be different, are called instances of the same Assembly. The Assembly Instance allows thus to create multiple instances of the same model hierarchy. An external Assembly can be plugged unmodified in another Assembly in the same way as with Russian Dolls. This flexibility for playing with Assemblies allows to increase their reusability, exchange and integration in simulators.

An Assembly Instance refers always to the Assembly it is an instance of. It appears as an instance node in the simulator hierarchy, this node corresponds then to the unique root Model Instance contained in the source Assembly. As such, each assembly instance in the Assembly can also be identified by a unique Path relative to the simulator top-level instance.

Each Assembly Instance contains:

a. A collection of Template Arguments defining the arguments applied to the source Assembly template parameters.

b. A collection of Component Configurations, which mean to override the configurations defined in the source assembly.

Each Assembly Instance can optionally be associated with a Configuration and/or a Link Base, which are meant to provide additional configuration or link information for that assembly instance.

4.2.2.4 Link

A Link defines a connection between two models. Following kinds of Link exist:

- A Field Link connects an Output Field to an Input Field.
- An Event Link connects an Event Source to an Event Sink.
- An Interface Link connects a Reference to an Interface provider that is a Component that implements the Interface. The Interface Link corresponds exactly to the Usage – Implementation pattern in UML. Furthermore, an Interface Link can be bidirectional meaning that there is a reverse interface link running from the client instance back to the owner instance (their respective roles in the link are reversed). This pattern is very useful to model for instance a full-duplex data communication interface such as an UART link or a Spacewire link as it allows to reduce two identical interface links to one single link between the components. Another use case concerns the simulation of the power interface: the forward interface link connects a voltage provider to a voltage consumer and the back interface link connects a current provider to a current consumer.

The three kinds of Link are illustrated in Figure 4-2.

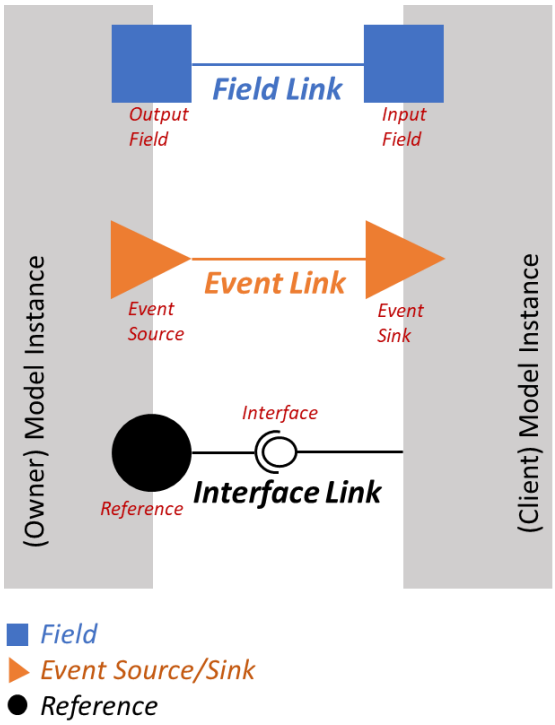

Figure 4-2: Three types of Link

In Figure 4-2, owner and client model instances are presented to express that Reference, Field, Event Source and Event Sink are sub-elements that belong always to a model instance. In reality, the link end objects can be found with their paths using the Resolver service, that allows to search for any SMP Object in the simulator hierarchy. In the information model, the three kinds of Link derive from Link, which has an Owner Path feature and a Client Path feature, which are:

- respectively the Output Field and Input Field paths in a Field Link.
- respectively the Event Source and Event Sink paths in an Event Link.
- respectively the Interface Consumer Model Instance and Interface Provider Model Instance paths in an Interface Link.

Figure 4-3 illustrates the bidirectional Interface Link:

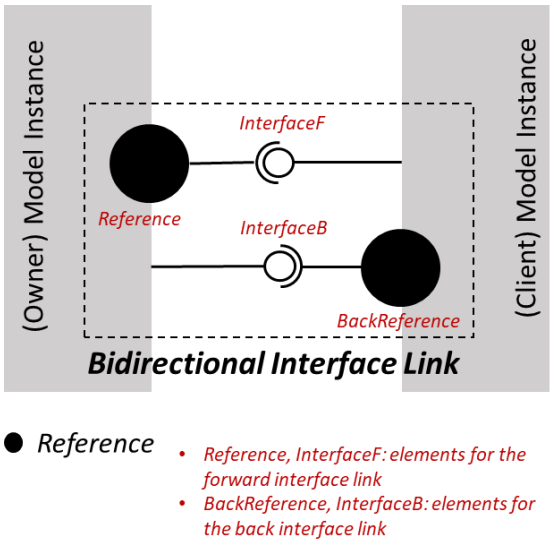

Figure 4-3: Bidirectional Interface Link

A bidirectional Interface Link models the communication in both the forward and the back directions. The *Reference* feature connects the owner instance to the client instance as a normal monodirectional link. If the link is bidirectional, the additional *BackReference* feature connects back the client instance, which becomes thus owner, to the owner instance, which becomes thus client.

A "bidirectional Field Link" can also be envisaged but there is currently no use case identified for such Link.

While a "bidirectional Event Link" cannot exist because due to the nature of an Event Link, attempting to model bidirectional communication for such Link does not really make sense.

4.2.2.5 Component Configuration

The Assembly and the Assembly Instance have a collection of Component Configurations as their direct children, which allows to override the configuration of the components that are direct or indirect children of the respective Assembly or Assembly Instance in the hierarchy. The Component Configuration element can be used to configure a Component, or a model instance, in the simulator, it contains the sub-elements shown in Figure 4-4.

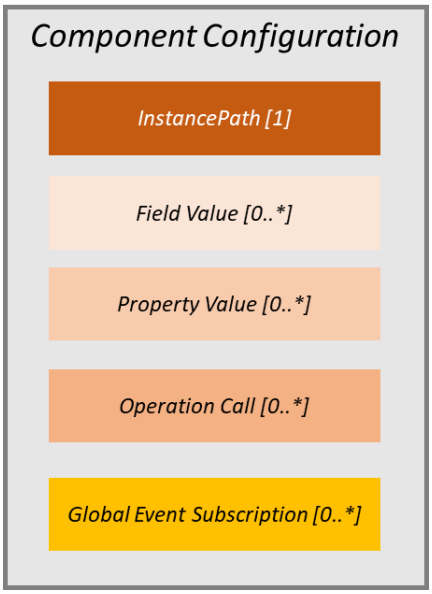

Figure 4-4: Component Configuration

In Figure 4-4, the following information is shown:

- Instance Path defines the path to the component that is being configured by the Component Configuration.
- The Field Value defines a value for a field belonging to the Component. See ECSS-E-ST-40-07 for the Field Value definition.
- The Property Value defines a value for a Property belonging to the Component.
- The Operation Call defines an operation belonging to the Component that is invoked for the configuration of the Component.
- The Global Event Subscription specifies which Entry Point from the Component needs to register with which Global Event for the configuration of the Component.

The Component Configurations that are parts of a top-level Assembly, which is an Assembly loaded at the simulator root level, can contain configuration for Model components as well as well as for Service components. In this latter case, the related Instance Paths are the absolute paths to the targeted Service components.

The Component Configurations that are parts of a non-top-level Assembly or of an Assembly Instance, contain only configurations for Model components. That is, the related Instance Paths are paths relative to the parent root nodes.

As Component Configuration can be used to deal with potentially a numerous quantity of configuration data, an initialisation order is necessary to ensure a coherent Component state. The initialisation order of elements found in a Component Configuration corresponds to the processing order during application of the Assembly artefact. Within elements of the same kind, the order corresponds to the order specified in the Component Configuration element.

As a configuration element in a Component Configuration (element can be Field Value, Property Value, Operation Call, Global Event Subscription) can be set as well in a Model Instance, it is important to specify an application order:

- First, the element found in the Model Instance is applied.
- Second, the element found in the Component Configuration, that is part of the Assembly and that has the instance path pointing to the same model instance, is applied.
- Last, the element found in the Component Configuration, that is part of the Assembly Instance created from the same Assembly and that has the instance path pointing to the same model instance, is applied.

Thus, a Component Configuration under an Assembly Instance is the last applied elements because its purpose is to override the successive configurations set for the related Model Instance and for the related Assembly (from which the Assembly Instance is created).

4.2.2.6 Template Argument

In a way analogue to a software function, an Assembly can define parameters for which values are assigned when the Assembly is instantiated. The parameters mean to allow some degree of freedom to customise the information found in the Assembly to the need of users (e.g. make a model instance name configurable). The parameters are associated with corresponding placeholders in the Assembly file, these respect a fixed syntax and are to be substituted with the parameter values at instantiation time.

Template Arguments allow to implement these parameters. In the current specification, Int32 and String8 Template Argument are supported. A Template Argument specifies a Parameter name and a Parameter default value. The placeholder is specified using the *{<parameter name>}* syntax. To include a literal *"{"* or *"}"* in the Assembly, escaping is done by putting double braces, that is *"{{"* or *"}}"*.

When the Assembly is instantiated (i.e. creation of an Assembly Instance), at the placeholder specified for a parameter, the *{<parameter name>}* string is substituted

by the actual argument value that is provided as part of the Assembly Instance specification.

Parameter substitution needs to be processed during loading and before the Assembly or the Assembly Instance information resulting from the substitution is actually applied to the simulator.

Figure 4-5 provides an example for the usage of Template Arguments.

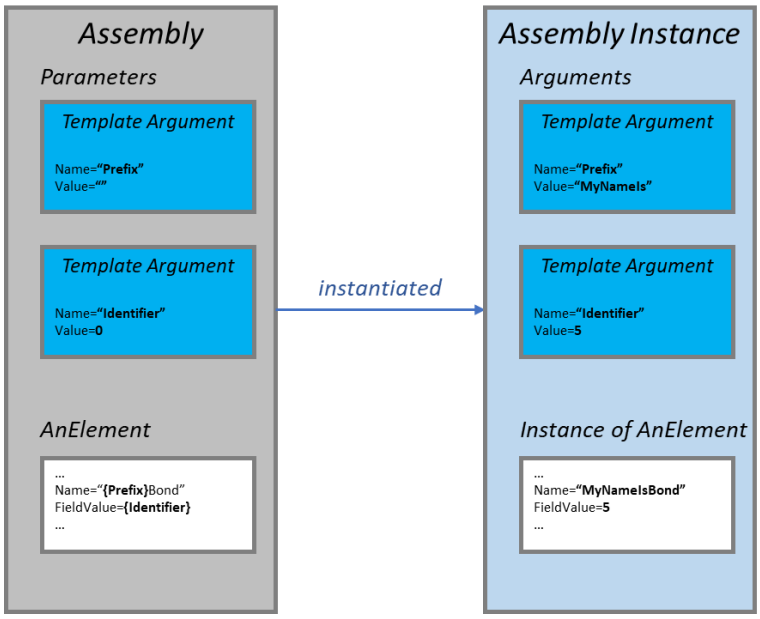

Figure 4-5: Example of Template Argument Usage

In the example shown in Figure 4-5, the *Prefix* and *Identifier* parameters are defined in the Assembly with their default values (respectively an empty string for *Prefix* and the value *"0"* for *Identifier*). In some element found in the Assembly, called here *AnElement* for illustration (e.g. *AnElement* can be *ModelInstance*), the placeholders are set as *{Prefix}* in the *AnElement* Name attribute and *{Identifier}* in the *AnElement* FieldValue attribute. When the Assembly is instantiated as an Assembly Instance with the provision of new Template Argument values, the placeholders are substituted with the argument values, that is *{Prefix}* with *"MyNameIs"* and *{Identifier}* with *"5"*, resulting in *AnElement* Name = *"MyNameIsBond"* and *AnElement* FieldValue = *"5"*. The resulting values *"MyNameIsBond"* and *"5"* are then sent for application to the simulator after parameter substitution is performed. In the simulator, *AnElement* is named *"MyNameIsBond"* and has a FieldValue equal to *"5"*. Note that if the Assembly Instance is created without specifying Template Argument values, then the placeholders are substituted with the default values from the Assembly. In this case, if *AnElement* is a model instance, the Name to be applied in the simulator would be *"Bond"* because the default value of the *Prefix* parameter is an empty string. This situation can raise a processing error for the Assembly if the final value for the *AnElement* Name is unexpected.

As a summary, Template Arguments are elements of the Assembly used at the file level. They are not functional building blocks of an assembly and are rather software elements, which need interpretation and substitution prior to the actual application to the simulator. This is offered as a mean to customise the Assembly file content with user-defined parameters.

4.2.2.7 Black-box Assembly

Well before the existence of ECSS-E-ST-40-08, Assemblies have been used quite successfully to configure simulators. The Assembly either followed the SMP2 format or was simply created using C++ code. After the SMP Level 1 standard, or ECSS-E-ST-40-07, is released, this practice using C++ code is still allowed because the SMP Level 1 standard does not cover the simulator assembly aspects. Thus, there is no reason to not allow the instantiation and configuration of simulators using solely C++ code. When used in conjunction with the ECSS-E-ST-40-08 Assembly artefact, the assembly provider can obviously specify the root model instance in the Assembly. Then at run time, the hierarchy of models below the root instance can be instantiated in many possible ways using C++ code:

- In the root model instance constructor.
- When the model Configure() is called.
- By calling a user operation that belongs to that root model instance: this is possible because user operations can be invoked when loading and applying an Assembly artefact.

The Assembly containing only one root model instance, which can afterwards create the full model hierarchy below it, is called a “full black-box assembly” because the whole content of the Assembly with the exception of the root instance is not visible to the model integrator, this latter having the task to assemble the simulation models to build-up the simulator.

Although possible, using full black-box assemblies has a serious issue in terms of simulator exchange as the model integrator does not know a priori how external models can connect to models contained in the assembly, neither which configuration values can be set in the assembly. Normally, this information is specified in the documentation that is delivered along with the Assembly (SMP Catalogues, models ICD…). Needless to say, this documentation is out of the scope of ECSS-E-ST-40-07 but at the end, using black-box assemblies is still not a good idea for simulator artefact exchange. Therefore, ECSS-E-ST-40-08 does not recommend this practice implicitly by not allowing the use of “full” black-box assemblies in simulator artefact exchange. Instead, ECSS-E-ST-40-08 allows the use of “partial” black-box assemblies by stating that when a model instance is required to connect to model instances external to the Assembly, meaning model instances found outside the perimeter of the hierarchy of models defined by the Assembly, that model instance needs to be created explicitly in the Assembly. Once specified, documentation about these model instances can be obtained from another source, for instance the corresponding SMP Catalogue or a model ICD, because these model instances refer to the corresponding models via their fully qualified type names.

4.2.3 Link Base Architecture

The Link Base is another useful optional artefact that can contribute to building the simulator. Basically, the Link Base contains a collection of Links (note that Link has been explained in 4.2.2.4) as shown in Figure 4-6.

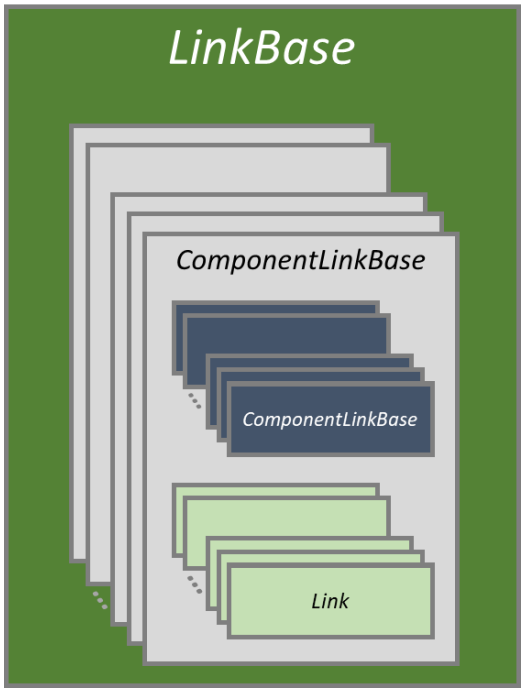

Figure 4-6: Link Base architecture

If used in a simulator, a Link Base can be loaded as a standalone artefact or it can also be loaded as an association with an Assembly Instance. In the first usage, the paths specified in the Link Base are paths in the simulator model hierarchy tree. In the second usage, the paths specified in the LinkBase are relative to the Assembly Instance the Link Base is associated with.

The Link Base can be used in some situation where there is the need to update the model connections in a simulator without modifying the model hierarchy and the corresponding configuration values. Typically, when the simulator has several top-level model instances, a Link Base can be used to create Links between them if needed because it is not allowed to add links in the sub-assemblies, which were served to instantiate those top-level instances. If used, the Link Base can be applied either during simulator startup or during the simulator Reconnecting state.

4.2.4 Schedule Architecture

4.2.4.1 General

The Schedule is an optional artefact that contributes to building up a simulator. It allows to schedule the events in the simulator scheduler to execute tasks. Following elements are the building blocks of a simulator schedule:

a. Zero or one Epoch Time to configure the Epoch time at the start of the simulation.

b. Zero or one Mission Start to configure the mission start time of the simulation.

c. Zero or more Tasks, each containing zero or more Activities.

d. Zero or more Events, each is associated with exactly one Task, which is the one that is executed by the Event when the Event occurs.

e. Zero or more Template Arguments, which work exactly like template arguments found in the Assembly (see description in 4.2.2 and 4.2.2.6).

Figure 4-7 shows the overview of a Schedule with all possible elements, which are contained therein:

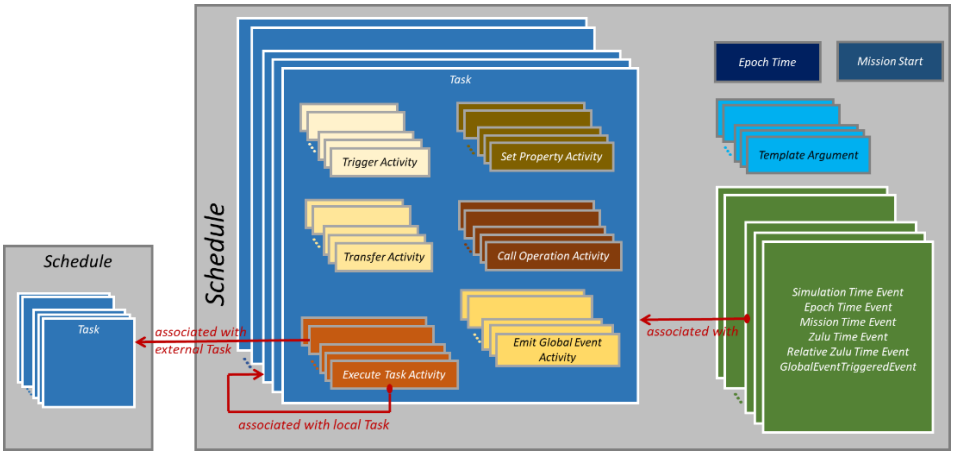

Figure 4-7: Schedule Architecture

The arrow "associated with" shown on Figure 4-7 means that an Event needs to reference the Task that it executes when occurred. An event can execute a Task defined in the same Schedule or a Task defined in an external Schedule via the "Execute Task Activity". If used, the Execute Task Activity can thus reference Tasks defined in the same Schedule ("associated with local Task" arrow) as well as defined in another Schedule ("associated with external Task" arrow). This referenced external Schedule can be considered as a schedule associated with a sub-assembly (see 4.2.2.3), which is typically associated with an equipment model. The "Execute Task Activity" offers a solution to reuse the Tasks defined in an external Schedule. On the other hand, it is not foreseen to reuse events defined in an external Schedule because this scheme creates an issue relative to the order of scheduling for events occurring at the same simulation date: if events were allowed to be specified in several schedules, additional semantics would be needed to handle the scheduling order for events occurring at the same

simulation date when these different schedules are applied to the simulator. Thus, a simulator is allowed to load and to apply at most one Schedule file containing at least one Event (without events, the Schedule would be useless!) and that single Schedule can possibly refer to Tasks defined in additional Schedules.

It is important to understand that the Schedule allows to plan execution of predictable Events for which occurrence times can be predicted at the start of a simulation. For instance, the event scheduled when an equipment is powered on is not predictable. Another term used for this kind of non-predictable event is "asynchronous" because it is impossible to know in advance from the start of a simulation when an equipment is turned on by the onboard software.

On the contrary of Events, Tasks are static information because they are linked to the simulation models which participate to the simulation and which are created via Assemblies from the simulation start.

4.2.4.2 Epoch Time

The Epoch time at simulation start can be configured using the Epoch Time element. See ECSS-E-ST-40-07 for Epoch time definition and handling details.

4.2.4.3 Mission Start

The mission start time of the simulation can be configured using the Mission Start element. See ECSS-E-ST-40-07 for mission time definition and handling details.

4.2.4.4 Template Argument

See 4.2.2.6 for details about Template Arguments.

Template Argument can be used to specify the root path in all the element paths that appear in the Activity definitions.

4.2.4.5 Task

A Task allows specifying the actions that can be performed in various ways when an event, which is associated with the Task and which is scheduled in the Scheduler, is executed. These various ways are defined through the Activities contained in the Task.

4.2.4.6 Various types of "Activity"

4.2.4.6.1 Activity

An Activity is a sub-element of, or contained in, a Task. The kind of an activity defines which actions are performed when the Activity is executed. The different activity kinds are defined in the following sub-clauses.

4.2.4.6.2 Trigger Activity

The Trigger Activity executes an Entry Point from a Model Instance. It is thus specified by the Resolver path to that related Entry Point.

4.2.4.6.3 Transfer Activity

The Transfer Activity performs the transfer of the Output field value to the connected Input field, these two are not necessary connected with each other via a Field Link. It is thus specified by the respectively Resolver paths to the Output and Input fields.

4.2.4.6.4 Set Property Activity

The Set Property Activity calls the setter of a Property from a Model Instance. It is thus specified by the Resolver path to the corresponding Property.

4.2.4.6.5 Call Operation Activity

The Call Operation Activity calls an Operation from a Model Instance. It is thus specified by the Resolver path to the corresponding Operation and by the collection of parameter values provided for the operation call.

4.2.4.6.6 Emit Global Event Activity

The Emit Global Event Activity emits a global event, that is an event managed by the Event Manager service. It is thus specified by the name of the global event plus an optional synchronous flag. See the definition in ECSS-E-ST-40-07.

4.2.4.6.7 Execute Task Activity

The Execute Task Activity execute a Task, which can be defined in the same Schedule, called a local Task in this first case, or in an external Schedule, called external Task in this second case. In this second case, as the external Schedule is usually associated with a sub-assembly, the Execute Task Activity needs two additional information to resolve the referenced Task:

- The Root Path pointing to the Assembly Instance that is created from the corresponding sub-assembly.
- A collection of Template Arguments in the case that those are used in the corresponding sub-assembly. The template argument values are needed to resolve by substitution the paths that are found in the activities contained in the Task.

4.2.4.7 Various types of “Events”

4.2.4.7.1 Event

An Event corresponds to an event scheduled for execution in the Scheduler service. Like defined in ECSS-E-ST-40-07, an event is defined by:

- A cycle time for a cyclic event or 0 for a one-shot event, which is an event that executes exactly one time.
- A repetition count, i.e. the number of occurrences of the event. For a one-shot event, the repetition count is 0.
- The time at which the event occurs for the first time: as there exist four kinds of time, which are Simulation, Epoch, Mission or Zulu time, this leads to four kinds of Event as well.

In addition to the Simulation, Epoch, Mission or Zulu event kinds, a new event kind is introduced by ECSS-E-ST-40-08: Global Event Triggered Event that triggers upon occurrence of a specific global event, which is a simulator-wise notification managed by the Event Manager service.

An Event is always associated with exactly one Task, that is executed each time that the event occurs.

Following sub-clauses provide details for the different kinds of event.

4.2.4.7.2 Simulation Time Event

The Simulation Time Event has its first occurrence time defined as a Simulation Time, i.e. a relative time since simulation starts.

4.2.4.7.3 Epoch Time Event

The Epoch Time Event has its first occurrence time defined as an Epoch Time, i.e. an absolute time. This kind of event depends on the Epoch time specified for time 0 of the simulation.

4.2.4.7.4 Mission Time Event

The Mission Time Event has its first occurrence time defined as a Mission Time, i.e. a relative time since the mission start time. This kind of event depends on the mission start time specified for the simulation.

4.2.4.7.5 Zulu Time Event

The Zulu Time Event has its first occurrence time defined as a Zulu Time, which corresponds to the absolute system time, or wall-clock time. This kind of event can obviously not keep its specification of time from one simulation to another.

4.2.4.7.6 Global Event Triggered Event

The Global Event Triggered Event is not triggered by a time, it is triggered by occurrence of a global event. Thus, this kind of event does not specify a first occurrence time. Instead, the event is triggered for the first time when a Start global event occurs and the event ceases to exist when a Stop global event occurs. This means that this kind of event is specified through a couple of Start and Stop global event names, which are defined and known to the Event Manager service. An example of use case of global event triggered event includes an OBSW mode change event, which triggers scheduling of a bunch of time events performed at the start global event. These scheduled time events can then be removed from the scheduler at the end of the event, which corresponds to the end of the OBSW mode. Especially, this scheme can be used to re-schedule a Zulu time event from the execution of a simulation to another. A global event triggered event can be scheduled to start or to end when the *Enter/LeaveExecuting* global events are triggered. When that event occurs, the related Zulu time event can be scheduled to fulfil the purpose.

4.2.5 Simulator Architecture

First of all, it is important to note that there is no artefact corresponding to the simulator architecture, which is rather composed of the simulator artefacts defined earlier, which are Assembly, Schedule and Link Base artefacts.

The simulator architecture excluding the simulation environment consists in a set of Assemblies, at least one needs to be provided, zero or one Schedule and zero or more Link Bases. If specified, the unique Schedule can use Tasks, never Events, defined in other Schedules. The simulator architecture and its relationships with simulator artefacts are illustrated in Figure 4-8.

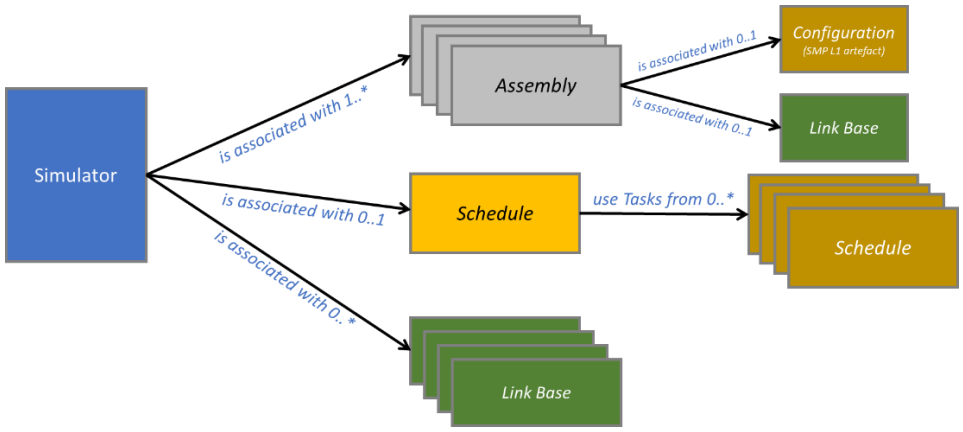

Figure 4-8: Simulator Architecture

Once loaded and applied, each Assembly results in one root Model Instance in the simulator hierarchy of Model Instances. One of them can be the Spacecraft Platform Assembly, others can be Assemblies modelling the Spacecraft Payloads or modelling supporting components of the spacecraft simulation. For example, there can be one Ground Segment root model instance containing Ground Station model instances. Alternatively, the simulator can be defined by one single Assembly in which Assembly Instances are created from external Assemblies. In this case, there is one single root Model Instance called for instance “Spacecraft”. Which simulator structure is followed is a choice that is made at design time during the spacecraft simulator development phase.

The hierarchical organization of a simulator, that is, of the corresponding Assemblies, reflects often the actual hardware components hierarchy of the real system that is being simulated.

Use of Schedules is optional and depends also on the choices made during the simulator design and integration. If used, exactly one Schedule is loaded and applied in the simulator. This unique schedule may use Tasks defined in other schedules, which can for example be associated with sub-assemblies. This means that the simulation infrastructure can load more than one Schedules but it applies only the one associated with the simulator and considers only the Tasks in external Schedules that are referred to by that applied Schedule.

Use of Link Bases, at the top-level or at the subassembly-level, is optional, it depends also on the choices made during the simulator design and integration.

A simulator can use as well zero or more SMP Level 1 Configuration artefacts. This last artefact has been specified in ECSS-E-ST-40-07 and is thus not defined in ECSS-E-ST-40-08 but as the main use of a Configuration artefact is with a simulator, this standard focuses mainly on its usage in harmony with the other simulator artefacts.

4.3 Simulator Metadata

4.3.1 General

The Assembly, Schedule and Link Base data models as presented in Clause 4.2 are stored in SMDL files, which are XML documents called respectively Assembly, Schedule and Link Base artefacts. These XML documents are compliant with the XML Schemas defined by ECSS-E-ST-40-08.

A simulator can thus be configured from its set of associated Assembly, Schedule and Link Base XML documents.

4.3.2 Assembly

Meta data for the Assembly elements is stored in an XML document called the Assembly. Having the Assembly elements defined in XML assemblies allows taking benefit from the XML language, for example to define the building blocks and data of a simulator in a highly structured language.

The XML document structure and content reflects exactly the data model defined in Clause 4.2.2. In the Assembly XML file, the Assembly root node corresponds to a Document root node, that contains:

- One root Model Instance node, which in turn contains
  - Zero or more (Sub-) Model Instance nodes (each can contain recursively (Sub-) Instance nodes). Note that the names in the metamodel, or in the XML data model, are slightly different than the ones used in this document:
    - XML element name is “ModelInstance” for the (Root) Model Instance
    - XML element name is “SubModelInstance” for the (Sub-) Model Instance
  - Zero or more Assembly Instance nodes (each can contain recursively Instance nodes)
  - Zero or more Link nodes
  - Zero or more Field Value nodes
  - Zero or more Property Value nodes
  - Zero or more Operation Call nodes
  - Zero or more Global Event Subscription nodes
- Zero or more Template Argument nodes

- Zero or more Component Configuration nodes, each containing in turn:
  - Zero or more Field Value nodes
  - Zero or more Property Value nodes
  - Zero or more Operation Call nodes
  - Zero or more Global Event Subscription nodes

4.3.3 Link Base

The XML document structure and content reflects exactly the data model defined in Clause 4.2.3. In the Link Base XML file, the Links root node corresponds to a Document root node, that contains one or more Component Link Base nodes, each contains one or more Link nodes.

4.3.4 Schedule

The XML document structure and content reflects exactly the data model defined in Clause 4.2.4. In the Schedule XML file, the Schedule root node corresponds to a Document root node, that contains zero or one Epoch Time node, zero or one Mission Start node, one or more Task nodes, each Task contains one or more Activity nodes, and one or more Event nodes among the four possible event kinds.

4.4 Simulator Initialisation and Configuration

4.4.1 General

This clause explains the principles that are at the basis of a standardised simulator initialisation sequence performed from the Assembly, Schedule, Link Base and Configuration artefacts, the latter being specified in the SMP Level 1 ECSS-E-ST-40-07 standard. It is critical to agree on a standardised simulator initialisation sequence in order to enhance portability in the exchange of the simulator artefacts, which can potentially be used in different simulation infrastructures.

To shorten the description, the generic term *"Value"* is used to designate a configuration element read from the Assembly. Thus, when mentioned, a *"Value"* can be one of the following:

- A Field Value
- A Property Value, i.e. a call to the corresponding Property setter
- An Operation Call
- A Global Event Subscription

The term *"Link"* has the same meaning as defined in 4.2.2.4.

4.4.2 Use Cases and Assumptions

4.4.2.1 General

Prior to tackling the simulator init sequence, it is important to analyse the use cases, which can be encountered when a simulator is initialised and configured from simulator artefacts. Some simplification assumptions need as well to be considered in order to standardise the artefact utilization, which suits all in practice for an effective simulator exchange scheme.

4.4.2.2 Compliance with the Early Simulator States

In order to build and to configure correctly a simulator starting from the set of Assembly, Schedule and Link Base files, the simulation infrastructure needs to load and apply them in accordance with the states of the simulation environment and in such a way that the configuration Values can be taken into account in the expected order to reach a coherent state of the simulator at the end of the initialisation phase. This ensures also that no false errors are detected in the simulator Assemblies, Schedules and Link Bases.

The simulation environment state machine from ECSS-E-ST-40-07 is reminded in Figure 4-9:

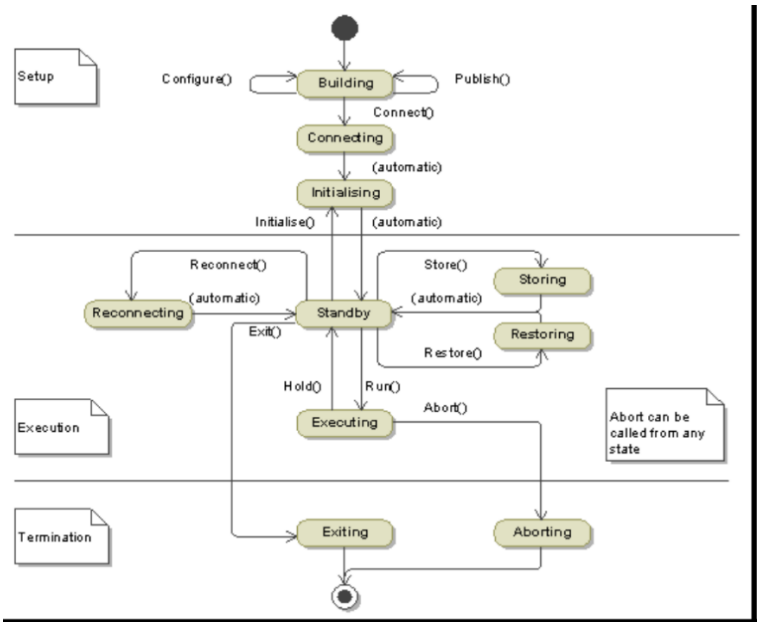

Figure 4-9: ECSS-E-ST-40-07 Simulation Environment state machine

Conformingly to ECSS-E-ST-40-07, it is possible to perform simulator instantiation and configuration from the artefacts while in the Building state.

4.4.2.3 Creation of new elements during simulator startup

In Clause 4.2.2.7, it has been stated that full black-box assemblies are not supported in ECSS-E-ST-40-08 for the purpose of simulator artefact exchange. If new model instances are created in C++ code during simulator startup, they are obviously not of interest in the model integration work. Such instances are not under management of the simulation infrastructure while it is loading and applying an Assembly. Thus, the full black-box assembly case can be ruled out in the simulator init sequence specification.

But the application of the Assemblies needs to be able to cope with the case where additional model instance sub-elements such as Fields, Properties, Event Source, Event Sinks, Entry Points, References, Operations are created dynamically during simulator startup. Those are usually child elements created by their parent model when the model Configure() method is called or when an Operation call specified in the Assembly is applied. Those elements do not exist yet at the Assembly loading time but their paths are well allowed in Links and Values configuration data. Thus, these paths are invalid, meaning that they cannot be resolved by the Resolver service, at the start of the assembly application but become valid after the new elements are created.

Furthermore, other artefacts like Link Base, Configuration and Schedule can be used together with Assemblies to configure a simulator. The case of non-existing elements in those artefacts can also be considered.

4.4.2.4 Assumption for Operation Calls

Correct behaviour of an operation called on a model requires a properly configured model state. Model internal configuration is achieved after the model Configure() is called while connection with the simulation environment can only be obtained after the model Connect() is called. E.g. in the called operation, the model might use the Scheduler service, which is possible only after the model Connect(). This means that a model can be considered as completely configured only after both the model Configure() and the model Connect() are called. In the most general case, it is thus expected that an operation call specified in the Assembly can be applied successfully only if this happens after the model Connect(). Supporting the general case makes however the implementation of an Assembly Loader more complex as it needs to defer the handling of Operation Calls until the Connecting state.

Therefore, a simplification assumption needs to be taken by stating that the operations called while processing Assembly concern only operations dedicated to the internal configuration of the called models. This means that these operations cannot use the simulation environment services other than the Logger service, which is available early to the model as it is already available for use from the model Configure() call onwards. Furthermore, it has to be noted that depending on the moment where a model Operation from an Assembly is applied, before or after the Configure() call, it can or cannot use the Logger service.

This rule cannot be enforced by the simulation infrastructure Assembly Loader because all this does is to apply the Operation Calls set in the provided Assembly. It has to be ensured of instead by the assembly provider. Call to operations, which change the model dynamic behaviour, can for example be specified instead in the Schedules which provide a better support for the model dynamic behaviour.

4.4.2.5 Assumption for Property Values

Handling of Property Values specified in an Assembly faces an ambiguity because there are two possible options for setting a Property:

- If the Property has an Attached Field, it is sufficient to set the Attached Field. This means that in this case, the Property Value can be handled like a Field Value.
- If the Property does not have an Attached Field, there is no other choice than calling the Property setter.

As ECSS-E-ST-40-07 does not define an operation in *Smp::IProperty* to determine whether a Property has an Attached Field or not, the decision on how to configure a Property is left with the provider of the Assembly, who has the knowledge of whether the Property has an Attached Field and who knows what the init order that allows reaching an overall coherent simulator state. The assembly provider can:

- Either decide to specify a Field Value when the property is attached to a field.
- Or decide to use a Property Value. Property Value found in an Assembly is always handled by calling the Property setter.
- Or even decide to use a Property Value in all cases for the sake of simplification.

4.4.3 Simulator Initialisation Process Description

The process described in this clause is based on the assumptions and use cases analyzed in Clause 4.4.2.

The simulator is initialized and configured entirely in the Building state from one or more top-level Assemblies, from zero or more Link Bases, from zero or more Configurations and from at most one Schedule (this Schedule can refer to Tasks defined in additional Schedules). The process is executed when calling methods defined in the *Smp::ISimulatorL2* interface (see [SMPL2_FILES]) for the purpose of loading and applying the various simulator artefacts:

- *LoadAssembly()* to load and to apply an Assembly file. This method can be called multiple times.
- *LoadLinkBase()* to load and to apply a Link Base file. This method can be called multiple times.
- *LoadConfiguration()* to load and to apply a Configuration file. This method can be called multiple times. Note that the SMDL Configuration is a file already specified in ECSS-E-ST-40-07 but there is no explicit method specified in *Smp::ISimulator* for the processing of that file. As a Configuration can be used in the initialization of a Level 2 simulator together with the other Level 2 artefacts, it is considered important to dedicate a specific method to load the Configuration file in the Level 2 specification.
- *LoadSchedule()* to load and to apply a Schedule file. This method can be called at most one time.

To provide full flexibility, these methods can be called in any order while in the simulator Building state.

LoadAssembly() allows and is able to handle elements that do not exist at loading time of the file. While LoadLinkBase(), LoadConfiguration() and LoadSchedule() allow non-existing elements in the respective processed files but it is not possible to resolve these during their application because there are no model Publish() calls or model Configure() calls during the processing of these files.

LoadAssembly() runs the process followed for loading and for applying an Assembly or a sub-assembly. Loading a sub-assembly requires in input the path to the parent model instance, the name of the container in which the assembly instance is created plus the assembly instance name that overrides the root model instance name in the sub-assembly. In the case of a top-level assembly, these parameters are respectively "/" (path to the model hierarchy root, a top-level assembly has no parent), "" (empty container name, a top-level assembly has no parent) and "" (empty assembly instance name, meaning that the assembly root model instance name is the name of the node created at the root of the simulator).

The Assembly loading starts in the Building state after all the model executable code as defined in Packages has been loaded.

During Assembly loading, the Template Arguments need to be processed first. They are to be applied immediately after reading in the corresponding parameters from the Assembly file and before that the values are effectively applied to the simulator. Template Arguments are processed by value substitution in order to create the final Assembly, which is applied to the simulator that is being initialized.

When Assembly loading is terminated, all Assembly objects specified in the file should be available in the computer memory. Subsequently, the application of the Assembly data happens in several steps:

a. All Model Instances are first created: this corresponds to the handling of the Model Instance and Assembly Instance elements from the loaded Assemblies.

b. Call *ISimulator.Publish()*.

c. Model Instances are then connected with each other: this corresponds to the handling of the Link elements from the loaded assemblies.

d. Unresolved Links resulting from model sub-elements that are created dynamically later (refer to the analysis given in Clause 4.4.2.3) are stored in a list, their application is deferred for later.

e. The Model Instance configuration via Component Configurations and Field Values from the loaded Assemblies are then applied: this is to ensure that model initialization performed during the model Configure() call can use the values specified in the Assemblies.

f. Unresolved Field Values (because of Fields which are created dynamically later – refer to the analysis given in Clause 4.4.2.3) are stored in a list, their application is deferred for later.

g. Next, the Model Instance configuration via Component Configurations, Property Values and Operation Call elements from the loaded Assemblies are applied.

h. Unresolved Property Values and Operation Calls (because of Properties/Operations which are created dynamically later – refer to the analysis given at Clause 4.4.2.3) are stored in a list, their application is deferred for later.

i. Last, the Global Event Subscriptions, the ones contained in Model Instance and the ones contained in Component Configurations, are processed.

j. Unresolved Entry Points (because of Entry Points which are created dynamically later – refer to the analysis given at Clause 4.4.2.3) are stored in a list, their application is deferred for later.

k. Call *ISimulator.Configure()*.

   1. This call traverses the model hierarchy and calls Configure() on each model instance. Before each model Configure() is called, the unresolved lists are checked to attempt resolving additional elements because new elements can have been created during the previous model Configure() call or a previous model Operation call or a Property setter call (see steps f. to j.).

   2. At the end of the simulator Configure() loop before exiting the call, resolution of elements is performed one more time to resolve remaining Links and Values.

Upon exit of the Configure() call, all unresolved elements are normally resolved because models are not allowed to create additional elements in the simulator Connecting state and because an Assembly is self-contained by design.

If used, LoadLinkBase() can be interleaved with the loading of the other artefacts. Unresolved links found in the Link Base are added to the corresponding list (see step d. above) for later processing in a subsequent LoadAssembly() call.

If used, LoadConfiguration() can be interleaved with the loading of the other artefacts. Unresolved field values found in the Configuration are added to the corresponding list (see step f. above) for later processing in a subsequent LoadAssembly() call.

If used, LoadSchedule(), which can be called only once, can be interleaved with the loading of other artefacts. Unresolved elements found in the Schedule are added to corresponding lists for later processing in a subsequent LoadAssembly() call. Note that Template Arguments can also be present in a Schedule. In that case, they are substituted during loading of a Schedule prior to the application of the Schedule data to the simulator.

The whole simulator initialization process can be visualized in Figure 4-10. To ease the process understanding, step numbering for LoadAssembly() is shown as well in the figure and is the same as in the description.

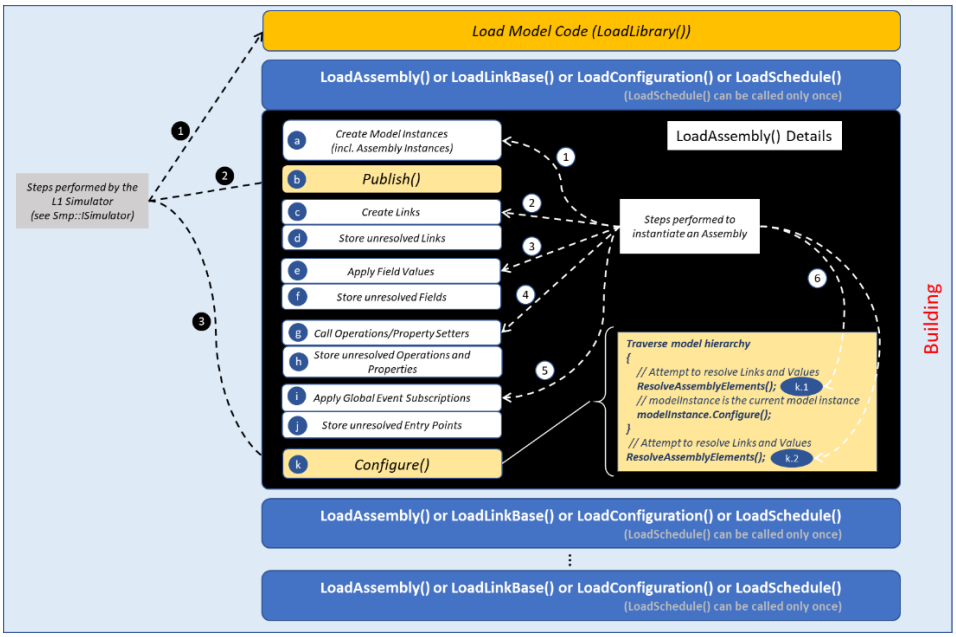

Figure 4-10: Simulator Initialisation

In Figure 4-10, the details of the steps for applying an Assembly are shown as well as the steps performed by the SMP Level 1 simulator. For the artefacts other than the Assembly, processing complexity is normally less than with the Assembly, hence details are not explicitly shown on the figure.

4.5 Simulator Reconfiguration

Simulator reconfiguration happens in the Reconnecting state. The Link Base, the Assembly, Schedule and Configuration artefacts can be applied to instantiate a new model hierarchy branch during the Standby state and then to connect it during the Reconnecting state. Loading and applying these artefacts to perform a reconfiguration happens exactly like specified in Clause 4.4.

5 Requirements

5.1 Common requirements

5.1.1 General

Assembly, Model Instance, Assembly Instance, Template Argument, Schedule, Task, Activity and Event have all a Name and a Description feature.

5.1.2 Requirements

ECSS-E-ST-40-08_1640001

a. The Name feature shall be a character string with the following features:
   1. Not be empty;
   2. Start with a letter;
   3. Contain only letters, digits, and underscore ("_");

ECSS-E-ST-40-08_1640002

b. The Description feature shall be a user free format character string.

ECSS-E-ST-40-08_1640003

c. The Name of an object belonging to a collection shall be unique in the context of that collection.

ECSS-E-ST-40-08_1640004

d. As an extension to 5.1.2c, names of Model Instances and Assembly Instances found at the same hierarchy level in different Containers shall be unique.

NOTE That is to say, even if these instances are created in different Containers, they have unique names.

ECSS-E-ST-40-08_1640005

e. Paths shall not contain "..".

NOTE Using ".." can allow potentially to navigate out of the model hierarchy defined within the current Assembly. Therefore, paths containing ".." are not allowed in an Assembly artefact.

5.2 Assembly requirements

5.2.1 Template Argument

5.2.1.1 General

Template Argument elements are Parameters for an Assembly and Arguments for an Assembly Instance. Note that the used terminology is the same as for a software function.

5.2.1.2 Requirements

ECSS-E-ST-40-08_1640006

a. A Template Argument shall be identified by a Name feature.

ECSS-E-ST-40-08_1640007

b. A Template Argument shall have a Description feature.

ECSS-E-ST-40-08_1640008

c. A Template Argument shall have a Value feature.

NOTE 1 When specified in an Assembly, a template argument defines its default value.

NOTE 2 When specified in an Assembly Instance, a template argument defines the actually applied value.

ECSS-E-ST-40-08_1640009

d. In the Assembly, the parameter placeholder shall be specified with the syntax *"{<parameter name>}"* where *<parameter name>* is equal to the corresponding Template Argument Name feature.

ECSS-E-ST-40-08_1640010

e. When the Assembly is instantiated, *"{<parameter name>}"* shall be substituted with the value specified as argument of the related Assembly Instance.

ECSS-E-ST-40-08_1640011

f. To escape the *"{"*, respectively *"}"*, literal *"{{"*, respectively *"}}"*, shall be used.

ECSS-E-ST-40-08_1640012

g. Integer32 Template Argument shall be supported.

ECSS-E-ST-40-08_1640013

h. String8 Template Argument shall be supported.

ECSS-E-ST-40-08_1640014

i. Template argument interpretation and substitution shall be performed during the loading of the Assembly and before applying the Assembly content to the simulator.

5.2.2 Field Value

5.2.2.1 General

A Field Value used in the configuration of a Model Instance contains the value of the field belonging to the model from which the model instance has been created. The Field Value specification details can be found in ECSS-E-ST-40-07.

A Field Value can be used to set the Attached Field of a Property at the place of a Property Value.

5.2.2.2 Requirements

ECSS-E-ST-40-08_1640015

a. Non-published Field found after unresolved elements processing or found Field with a different value type than the published one shall result in an error in the loading of the Assembly, which specifies the Field Value.

NOTE Unresolved elements processing is described in clause 5.5.1.2

5.2.3 Property Value

5.2.3.1 General

For the configuration of a Model Instance, a Property Value contains the value of the property belonging to the model from which the model instance has been created. Property Values can exist only for Read-Write or Write-Only properties (see the related definitions in ECSS-E-ST-40-07).

5.2.3.2 Requirements

ECSS-E-ST-40-08_1640016

a. Property Value shall provide the Property feature that contains the name of the corresponding property in the parent model instance.

ECSS-E-ST-40-08_1640017

b. Property Value shall provide the Value feature that contains the value to set for the corresponding property in the parent model instance.

ECSS-E-ST-40-08_1640018

c. Setting the Property Value in the Assembly shall invoke the corresponding Property setter operation.

ECSS-E-ST-40-08_1640019

d. Setting Property Value for a property that is Read-Only shall result in an error in the application of the Assembly, which specifies that Property Value.

NOTE The property access kind and type can be checked against the property published information and the type information stored in the Type Registry, when loading the Assembly.

ECSS-E-ST-40-08_1640020

e. Non-published Property (found after unresolved elements processing – see clause 5.5.1.2) or found Property with a different value type than the published one shall result in an error in the loading of the Assembly which specifies the Property Value.

5.2.4 Operation Call

5.2.4.1 General

For the configuration of a Model Instance, the Operation Call element allows invoking an Operation provided by that Model Instance.

5.2.4.2 Requirements

ECSS-E-ST-40-08_1640021

a. Operation Call shall provide the Operation feature that contains the name of the corresponding operation in the parent model instance.

ECSS-E-ST-40-08_1640022

b. Operation Call shall contain a collection of ordered Parameter Values of the corresponding operation.

NOTE "first appearance in collection" is equal to "first appearance in the operation signature".

ECSS-E-ST-40-08_1640023

c. Each Parameter Value shall provide the Name feature that identifies the name of the corresponding parameter of the corresponding operation.

ECSS-E-ST-40-08_1640024

d. Each Parameter Value shall provide the Value feature that contains the simple value of the corresponding parameter of the corresponding operation.

ECSS-E-ST-40-08_1640025

e. A return type Parameter Value shall be interpreted as the expected return value when applying the Operation Call.

ECSS-E-ST-40-08_1640026

f. When the Operation with a return value is called, the actual return value shall be compared with the return type Parameter Value to validate the success/failure status of the Operation Call.

ECSS-E-ST-40-08_1640027

g. Processing the Operation Call in the Assembly shall use *Smp::IDynamicInvocation* to invoke the corresponding Operation that belongs to the parent Model Instance.

NOTE See ECSS-E-ST-40-07 for details about *Smp::IDynamicInvocation*

ECSS-E-ST-40-08_1640028

h. Non-published Operation (found after unresolved elements processing – see clause 5.5.1.2) or found Operation with a different signature than the specified one shall result in an error in the loading of the Assembly which specifies the Operation Call.

NOTE 1 The operation existence or the operation signature can be checked against the operation published information and the type information stored in the Type Registry, when loading the Assembly.

NOTE 2 An operation is not published if it contains complex type parameters. Call to such Operation can also not be specified in an Assembly.

5.2.5 Global Event Subscription

5.2.5.1 General

The Global Event Subscription element allows registering an Entry Point belonging to a Model Instance with a global event, which is a simulator-wide event managed by the Event Manager service. The Model Instance is the parent model instance if the element is placed under that Model Instance or the Model Instance pointed to by the *InstancePath* attribute if it is placed under a Component Configuration

5.2.5.2 Requirements

ECSS-E-ST-40-08_1640029

a. The Global Event Subscription shall provide the *EntryPointName* feature that specifies the name of the Entry Point to be registered with the global event.

ECSS-E-ST-40-08_1640030

b. The Global Event Subscription shall provide the *GlobalEventName* feature that specifies the name of the global event to register the Entry Point with.

NOTE The global event is either a predefined global event or a new global event that exists or is to be created in the Event Manager service - see ECSS-E-ST-40-07 for details

ECSS-E-ST-40-08_1640031

c. An Entry Point name unknown to the parent Model Instance (found after unresolved elements processing – see clause 5.5.1.2) shall result in an error in the application of the Assembly which contains the Global Event Subscription.

NOTE The “Global Event name unknown to the Event Manager service” cannot be an error case because ECSS-E-ST-40-07 specifies that a non-existing global event can always be created at the first subscription.

5.2.6 Component Configuration

5.2.6.1 General

A Component Configuration can be defined for any Assembly or Assembly Instance in the hierarchy. It contains a collection of Field Values, Property Values, Operation Calls, Global Event Subscriptions plus the path to the Model Instance that owns those. Component Configuration is a mean in the Assembly or Assembly Instance to group all configuration elements that belong to the same component.

5.2.6.2 Requirements

ECSS-E-ST-40-08_1640032

a. The Component Configuration shall be defined as child element of an Assembly or of an Assembly Instance.

ECSS-E-ST-40-08_1640033

b. A Component Configuration shall provide the *InstancePath* feature that contains the relative Path with regards to the parent Assembly or Assembly Instance.

NOTE 1 For an Assembly Instance, the path is relative to the root Model Instance.

NOTE 2 The relative path does not include the Assembly Instance name or the Assembly root model instance name.

ECSS-E-ST-40-08_1640034

c. A Component Configuration may contain Field Values of the fields belonging to the component instance defined by the *InstancePath* feature.

ECSS-E-ST-40-08_1640035

d. A Component Configuration may contain Property Values of the properties belonging to the component instance defined by the *InstancePath* feature.

ECSS-E-ST-40-08_1640036

e. A Component Configuration may contain Operation Calls belonging to the component instance defined by the *InstancePath* feature.

ECSS-E-ST-40-08_1640037

f. A Component Configuration may contain Global Event Subscriptions relative to the component instance defined by the *InstancePath* feature.

5.2.7 Link

5.2.7.1 General

ECSS-E-ST-40-08_1640038

a. The Link shall be a child of a Model Instance.

NOTE Place of a Link can be found anywhere in the hierarchy below an instance but it is preferably specified directly under the root Model Instance in the Assembly. This use has the advantage to group all Links at one predefined place to allow increasing readability of the Assembly.

ECSS-E-ST-40-08_1640039

b. The Link shall provide an Owner Path feature that contains the path to the Object being the Link source.

NOTE The owner object is the object from which the link originates. It is a model instance path in the case of an Interface Link or otherwise a model instance sub-element path that is necessarily an output field or an event source.

ECSS-E-ST-40-08_1640040

c. The Link shall provide a Client Path feature that contains the path to the Object being the Link target.

NOTE The client object is the object in which the link ends. It is a model instance path in the case of an Interface Link or otherwise a model instance sub-element path that is necessarily an input field or an event sink.

ECSS-E-ST-40-08_1640041

d. A Link shall be either an Interface Link, an Event Link or a Field Link.

ECSS-E-ST-40-08_1640042

e. The Client Path shall refer to either an object in the current model instance or an object found in its children.

ECSS-E-ST-40-08_1640043

f. The Owner Path shall refer to either an object in the current model instance or an object found in its children.

5.2.7.2 Interface Link

ECSS-E-ST-40-08_1640044

a. The Interface Link shall connect a Reference with a Model Instance that implements the Interface that corresponds to the type of the Reference.

NOTE 1 Interface consumer and provider Model Instance are reached thanks to respectively the link Owner Path and the link Client Path

NOTE 2 Reference and Interface can be defined in SMP Catalogues (see ECSS-E-ST-40-07).

ECSS-E-ST-40-08_1640045

b. The Reference of the Owner Model Instance shall be identified by its Name.

ECSS-E-ST-40-08_1640046

c. The Interface Link may provide the BackReference feature that allows connecting back the Client Reference to the Owner Model Instance.

NOTE That is to say, the Interface Link is bidirectional and the participating model instance roles are reversed for the BackReference when it is present.

ECSS-E-ST-40-08_1640047

d. When BackReference exists, the Owner Model Instance shall implement the interface that corresponds to the type of the BackReference reference.

ECSS-E-ST-40-08_1640048

e. The BackReference of the Client Model Instance shall be identified by its Name.

ECSS-E-ST-40-08_1640049

f. Mismatch between the Interface implemented by the client model instance and the type of the Reference shall result in an error in the application of the Assembly which contains the Interface Link.

ECSS-E-ST-40-08_1640050

g. Mismatch between the Interface implemented by the owner model instance and the type of the BackReference shall result in an error in the application of the Assembly which contains the Interface Link.

5.2.7.3 Event Link

ECSS-E-ST-40-08_1640051

a. The Event Link shall connect an Event Source to an Event Sink, both being of the same Event Type.

NOTE Event source and sink are reached thanks to respectively the link Owner Path and the link Client Path

ECSS-E-ST-40-08_1640052

b. Mismatch in the Event Type between the event source and the event sink shall result in an error in the application of the Assembly which contains the Event Link.

NOTE The Event Type is defined in a SMP Catalogue (see ECSS-E-ST-40-07).

5.2.7.4 Field Link

ECSS-E-ST-40-08_1640053

a. The Field Link shall connect an Output Field to an Input Field.

NOTE Output and input fields are reached thanks to respectively the link Owner Path and the link Client Path

ECSS-E-ST-40-08_1640054

b. The Field Link shall allow connecting a sub-Field from a Structure Field.

ECSS-E-ST-40-08_1640055

c. If the Input or Output Field of a Field Link is a sub-Field of a Structure Field, the syntax <*StructureFieldName*>/<*subFieldName*> shall be used in the respective Client or Owner Path.

ECSS-E-ST-40-08_1640056

d. The Field Link shall allow connecting an Array Item from an Array Field.

ECSS-E-ST-40-08_1640057

e. If the Input or Output Field of a Field Link is an Array Item at the *index* position from an Array Field, the syntax <*ArrayFieldName*>*[index]* shall be used in the respective Client and Owner Path.

ECSS-E-ST-40-08_1640058

f. Non-strict compatibility or non-semantically equivalence between the output field and the input field shall result in an error in the application of the Assembly which contains the Field Link.

NOTE See strict compatibility and semantic equivalence definitions in ECSS-E-ST-40-07.

5.2.8 Model Instance

ECSS-E-ST-40-08_1640059

a. The Model Instance shall have a Name.

ECSS-E-ST-40-08_1640060

b. The Model Instance shall have a Description.

ECSS-E-ST-40-08_1640061

c. The Model Instance shall provide the *Implementation* feature, which contains either the fully qualified type name or the implementation UUID of the Model of which it is an instance.

NOTE 1 The fully qualified type name is the one of the Model C++ class.

NOTE 2 The Implementation UUID is as specified in ECSS-E-ST-40-07.

ECSS-E-ST-40-08_1640062

d. The Model Instance may contain Field Values.

ECSS-E-ST-40-08_1640063

e. The Model Instance may contain Property Values.

ECSS-E-ST-40-08_1640064

f. The Model Instance may contain Operation Calls.

ECSS-E-ST-40-08_1640065

g. The Model Instance may contain Global Event Subscriptions.

ECSS-E-ST-40-08_1640066

h. The Model Instance may contain Links.

ECSS-E-ST-40-08_1640067

i. A Link belonging to a Model Instance shall connect two instances, Model Instance or Assembly Instance, one of which can be the current instance or both can be found in the hierarchy below that Model Instance.

ECSS-E-ST-40-08_1640068

j. The Model Instance may contain child Model Instances.

NOTE The Model Instance is able to contain child Model Instances if it is an instance of a Model that is a Composite, implying that it has Containers.

ECSS-E-ST-40-08_1640069

k. The Model Instance may contain child Assembly Instances.

NOTE The Model Instance is able to contain child Assembly Instances if it is an instance of a Model that is a Composite, implying that it has Containers.

ECSS-E-ST-40-08_1640070

l. A Model Instance, which is not the root instance of an Assembly, shall have the Container feature, which provides the Name of the parent Model Instance container in which the model instance is created.

NOTE A child instance is added to one of the compatible Containers of the parent instance,

meaning it is the container with the object type that is compatible with the child instance type (same implemented Interface).

5.2.9 Assembly Instance (or Sub-Assembly)

ECSS-E-ST-40-08_1640071

a. The Assembly Instance, or Sub-Assembly, shall have a Name.

ECSS-E-ST-40-08_1640072

b. The Assembly Instance, or Sub-Assembly, shall have a Description.

ECSS-E-ST-40-08_1640073

c. The Assembly Instance, or Sub-Assembly, shall provide the Assembly feature that contains the SMDL file name of the Assembly of which it is an instance.

ECSS-E-ST-40-08_1640074

d. The hierarchy of model instances defined in the Assembly shall be inserted at the place where the Assembly Instance is created and with the Assembly Instance taking the place of the Assembly root instance.

ECSS-E-ST-40-08_1640075

e. The assembly instance Name shall override the Name of the root model instance found in the referenced Assembly.

NOTE In other words, the name of the node in the simulator hierarchy is the Assembly Instance name, not the referred Assembly root instance name.

ECSS-E-ST-40-08_1640076

f. The Assembly Instance, or Sub-Assembly, shall provide the Container feature containing the Name of the container in the parent Model Instance in which it is instantiated.

ECSS-E-ST-40-08_1640077

g. The Assembly Instance, or Sub-Assembly, may provide the Configuration feature that contains the file name of an associated SMDL Configuration file.

NOTE SMDL Configuration file details are specified in ECSS-E-ST-40-07

ECSS-E-ST-40-08_1640078

h. The SMDL Configuration file given by the Configuration feature shall be loaded and applied after the Assembly has been instantiated.

ECSS-E-ST-40-08_1640079

i. The Assembly Instance, or Sub-Assembly, may contain a collection of Component Configurations.

ECSS-E-ST-40-08_1640080

j. The Assembly Instance, or Sub-Assembly, may contain a collection of Template Arguments.

ECSS-E-ST-40-08_1640081

k. The Assembly Instance, or Sub-Assembly, may provide the LinkBase feature that contains the file name of an associated SMDL Link Base file.

NOTE SMDL Link Base file details are specified in clause 5.3.

ECSS-E-ST-40-08_1640082

l. The SMDL LinkBase file given by the LinkBase feature shall be loaded and applied after the Assembly has been instantiated.

ECSS-E-ST-40-08_1640083

m. The links defined in an associated Link Base file shall not override the links that exist already in the associated Assembly.

NOTE While overriding a field configuration is completely meaningful, this notion for a link does not make sense.

5.2.10 Assembly

ECSS-E-ST-40-08_1640084

a. The Assembly shall contain exactly one standalone root Model Instance.

NOTE "standalone Model Instance" means that the root model instance of the Assembly does not become yet a child of another Model Instance.

ECSS-E-ST-40-08_1640085

b. The unique root Model Instance found in the Assembly shall not have a Container feature.

NOTE This is a consequence of the Assembly being a standalone Model Instance as the Container can exist only in a parent instance.

ECSS-E-ST-40-08_1640086

c. The Assembly may contain a collection of Links.

ECSS-E-ST-40-08_1640087

d. The Assembly may contain a collection of Component Configurations.

ECSS-E-ST-40-08_1640088

e. The Assembly may contain a collection of Template Arguments.

5.3 Link Base requirements

ECSS-E-ST-40-08_1640089

a. The Link Base shall contain a collection of Component Link Bases.

ECSS-E-ST-40-08_1640090

b. The Component Link Base shall provide a Path feature that contains the path to a model instance existing in the context of the Link Base.

NOTE The context of the Link Base is defined at the moment that the Link Base artefact is loaded in the simulator either as a standalone artefact or as an association with an Assembly Instance.

ECSS-E-ST-40-08_1640091

c. The Component Link Base shall contain a collection of Links.

NOTE Refer to Clause 5.2.7 for requirements about the Link.

ECSS-E-ST-40-08_1640092

d. The collection shall contain at least one Link.

NOTE Otherwise, the Link Base artefact does not contain useful data.

ECSS-E-ST-40-08_1640093

e. The Component Link Base may contain a collection of Component Link Bases.

NOTE This allows a hierarchical organisation of the Component Link Bases contained in the Link Base.

5.4 Schedule requirements

5.4.1 Epoch Time

ECSS-E-ST-40-08_1640094

a. The Schedule may initialize the Epoch Time at the simulation start.

NOTE Epoch Time is defined in ECSS-E-ST-40-07

5.4.2 Mission Start

ECSS-E-ST-40-08_1640095

a. The Schedule may specify the Mission Time start at the simulation start.

NOTE Mission Start is defined in ECSS-E-ST-40-07

5.4.3 Template Argument

5.4.3.1 General

Requirements for Template Arguments defined for the Assembly in Clause 5.2.1 apply also to Schedule.

5.4.3.2 Requirements

ECSS-E-ST-40-08_1640096

a. Template argument interpretation and substitution shall be performed during the loading of the Schedule and before applying the Schedule content to the simulator.

ECSS-E-ST-40-08_1640097

b. At least one String8 Template Argument shall be used to specify the root path relative to which all element paths in the Schedule are interpreted.

5.4.4 Task

ECSS-E-ST-40-08_1640098

a. Task shall have a Name.

ECSS-E-ST-40-08_1640099

b. Task shall have a Description.

ECSS-E-ST-40-08_1640100

c. Task shall contain a collection of Activity.

NOTE An Activity type is one of the following: Trigger, Transfer, Set Property, Call Operation, Emit Global Event, Execute Task.

5.4.5 Activity

ECSS-E-ST-40-08_1640101

a. An Activity shall have a Name.

ECSS-E-ST-40-08_1640102

b. An Activity shall have a Description.

ECSS-E-ST-40-08_1640103

c. Trigger Activity shall provide the *EntryPoint* feature that contains the path to the Entry Point that the activity triggers when it executes.

ECSS-E-ST-40-08_1640104

d. Transfer Activity shall provide the *OutputFieldPath* feature that contains the path to the Output Field of which the activity transfers the value when it executes.

ECSS-E-ST-40-08_1640105

e. Transfer Activity shall provide the *InputFieldPath* feature that contains the path to the Input Field to which the activity transfers the value when it executes.

ECSS-E-ST-40-08_1640106

f. Emit Global Event Activity shall provide the *EventName* feature that contains the name of the global event to emit when the activity executes.

ECSS-E-ST-40-08_1640107

g. Emit Global Event Activity may provide the *synchronous* feature (Boolean flag) that defines the flag associated with the global event emission.

NOTE See ECSS-E-ST-40-07 for details regarding this flag.

ECSS-E-ST-40-08_1640108

h. In absence of the *synchronous* feature, its default value shall be as specified in ECSS-E-ST-40-07.

ECSS-E-ST-40-08_1640109

i. Set Property Activity shall provide the *PropertyPath* feature that contains the path to the Property of which the activity sets the value when it executes.

ECSS-E-ST-40-08_1640110

j. Call Operation Activity shall provide the *OperationPath* feature that contains the path to the Operation that the activity calls when it executes.

ECSS-E-ST-40-08_1640111

k. Execute Task Activity shall be associated with exactly one Task, which can be defined in the same or in a different Schedule, that the activity executes.

ECSS-E-ST-40-08_1640112

l. Execute Task Activity may provide the *RootPath* feature that contains the root model instance path of the sub-assembly with which the referenced external Schedule is associated.

ECSS-E-ST-40-08_1640113

m. Absence of *RootPath* shall default the root model instance path to the simulator root path

NOTE Simulator root path is "/".

ECSS-E-ST-40-08_1640114

n. Execute Task Activity may contain a collection of *TemplateArguments* to resolve the ones found in the referenced external Schedule.

5.4.6 Event

ECSS-E-ST-40-08_1640115

a. An Event shall have a Name.

ECSS-E-ST-40-08_1640116

b. An Event shall have a Description.

ECSS-E-ST-40-08_1640117

c. An Event shall provide the *CycleTime* feature that specifies the event cycle duration.

NOTE Semantics of this feature is as specified in ECSS-E-ST-40-07.

ECSS-E-ST-40-08_1640118

d. An Event may provide the *RepeatCount* feature that specifies the number of repetitions of the event occurrence.

NOTE Semantics of this feature is as specified in ECSS-E-ST-40-07.

ECSS-E-ST-40-08_1640119

e. Simulation Time Event shall provide the *SimulationTime* feature that specifies the event first occurrence simulation time.

NOTE Semantics of this feature is as specified in ECSS-E-ST-40-07.

ECSS-E-ST-40-08_1640120

f. Epoch Time Event shall provide the *EpochTime* feature that specifies the event first occurrence Epoch time.

NOTE Semantics of this feature is as specified in ECSS-E-ST-40-07.

ECSS-E-ST-40-08_1640121

g. Mission Time Event shall provide the *MissionTime* feature that specifies the event first occurrence mission time.

NOTE Semantics of this feature is as specified in ECSS-E-ST-40-07.

ECSS-E-ST-40-08_1640122

h. Zulu Time Event shall provide the *ZuluTime* feature that specifies the event first occurrence Zulu time.

NOTE Semantics of this feature is as specified in ECSS-E-ST-40-07.

ECSS-E-ST-40-08_1640123

i. Global Event Triggered Event shall provide the *StartEvent* feature that contains the name of the global event that, when emitted, triggers the start of existence of the event.

ECSS-E-ST-40-08_1640124

j. Global Event Triggered Event may provide the *StopEvent* feature that contains the name of the global event that, when emitted, triggers the end of existence of the event.

ECSS-E-ST-40-08_1640125

k. Global Event Triggered Event may provide the *TimeKind* feature that contains the time kind to use for the Event, which is one of the four available SMP Time Kinds.

NOTE SMP time kinds are specified in ECSS-E-ST-40-07.

ECSS-E-ST-40-08_1640126

l. Global Event Triggered Event may provide the *Delay* feature that contains the delay after which the cyclic events scheduled from *StartEvent* are executed the first time.

ECSS-E-ST-40-08_1640127

m. An Event shall be associated with exactly one Task defined in the same Schedule.

NOTE An Event can reach an external Task via the Execute Task Activity.

ECSS-E-ST-40-08_1640128

n. An Event shall execute the associated Task when the event occurs.

5.5 Simulator initialisation and configuration

5.5.1 Simulator initialisation and configuration with Assemblies, Link Bases and Configurations

5.5.1.1 General

To ease understanding, the requirement numbering is identical to the step numbering used in Clause 4.4.3.

The *simulator Publish() call* refers to the *Smp::ISimulator::Publish() method* as specified in ECSS-E-ST-40-07.

The *simulator Configure() call* refers to the *Smp::ISimulator::Configure() method* as specified in ECSS-E-ST-40-07.

A *model Configure() call* refers to the *Smp::IComponent::Configure() method* as specified in ECSS-E-ST-40-07.

5.5.1.2 Requirements

ECSS-E-ST-40-08_1640129

a. First, all Model Instances, regular Model Instances as well as Assembly Instances, specified in Assemblies shall be created.

ECSS-E-ST-40-08_1640130

b. The simulator Publish() method shall be called.

ECSS-E-ST-40-08_1640131

c. Links specified in Assemblies and in Link Bases shall then be created.

NOTE The Publish() call allows elements appearing in Links to be published and thus become visible to the simulation environment so that it can later resolve and create successfully the corresponding Links.

ECSS-E-ST-40-08_1640132

d. Unresolved Links, which cannot be created in 5.5.1.2c, shall be stored in UNRESOLVED_LINKS.

NOTE 1 They are memorized in order to be resolved later in the process.

NOTE 2 UNRESOLVED_LINKS is the name of a variable used in the implementation for the purpose of memorizing the unresolved links.

ECSS-E-ST-40-08_1640133

e. The field values, including the values which are part of Component Configuration elements, specified in all Assemblies shall be applied after the Links creation in the following order:

1. The field values under a Model Instance;
2. The field values under a Component Configuration in the same Assembly pointing to the same model instance;
3. The field values under a Component Configuration that is part of the Assembly Instance.

NOTE The models can use the field values specified in Assemblies to perform internal configuration (see requirements related to the model Configure() call).

ECSS-E-ST-40-08_1640134

f. Unresolved field values, which cannot be applied in 5.5.1.2e, shall be stored in UNRESOLVED_FIELD_VALUES.

NOTE 1 They are memorized in order to be resolved later in the process.

NOTE 2 UNRESOLVED_FIELD_VALUES is the name of a variable used in the implementation for the purpose of memorizing the unresolved field values.

ECSS-E-ST-40-08_1640135

g. The operation calls and property values, these latter correspond to the property setters, specified in all Assemblies shall be applied after the field values in the following order:

1. The operation calls and property values under a Model Instance;
2. The operation calls and property values under a Component Configuration in the same Assembly pointing to the same model instance;
3. The operation calls and property values under a Component Configuration that is part of the Assembly Instance.

NOTE The models can use the configuration data provided by these elements to perform internal configuration (see requirements related to the model Configure() call).

ECSS-E-ST-40-08_1640136

h. Unresolved operation calls and property values, which cannot be applied in 5.5.1.2g, shall be stored in UNRESOLVED_OPERATIONS.

NOTE 1 They are memorized in order to be resolved later in the process.

NOTE 2 UNRESOLVED_OPERATIONS isreplaced by the name of a variable used in the implementation for the purpose of memorizing the unresolved property values and operation calls.

ECSS-E-ST-40-08_1640137

i. The Global Event Subscriptions, including the ones that are part of Component Configurations, specified in all Assemblies shall be applied after the Operation Calls and Property Values in the following order:

1. The global event subscriptions under a Model Instance;
2. The global event subscriptions under a Component Configuration in the same Assembly pointing to the same model instance;
3. The global event subscriptions under a Component Configuration that is part of the Assembly Instance.

NOTE Order of subscription to global events is not important.

ECSS-E-ST-40-08_1640138

j. Unresolved global event subscriptions, which cannot be applied in 5.5.1.2i, shall be stored in UNRESOLVED_GEVENT_SUBSCRIPTIONS.

NOTE 1 They are memorized in order to be resolved later in the process.

NOTE 2 UNRESOLVED_GEVENT_SUBSCRIPTIONS is replaced by the name of a variable used in the implementation for the purpose of memorizing the unresolved global event subscriptions.

ECSS-E-ST-40-08_1640139

k. When called, the simulator Configure() method shall attempt to resolve elements in UNRESOLVED_LINKS, UNRESOLVED_FIELD_VALUES, UNRESOLVED_OPERATIONS and UNRESOLVED_GEVENT_SUBSCRIPTIONS as follows:

   1. Each time before calling a model Configure()method;
   2. After all model Configure() methods have been called.

ECSS-E-ST-40-08_1640140

l. Unresolved elements left at the end of the init sequence shall raise an Assembly application error.

ECSS-E-ST-40-08_1640141

m. The application order of field values, property values and operation calls shall follow their specification order in the Assembly or in the Link Base artefact.

   NOTE The specification order is in general the order in which the loader reads in from the file: elements at the top are read-in first, the loader progressing towards the bottom of the file.

ECSS-E-ST-40-08_1640142

n. Adding elements to unresolved element lists shall respect the order in which the elements appear in the Assembly.

ECSS-E-ST-40-08_1640143

o. The loading and application of a Configuration artefact may be interleaved with the loading and application of the other simulator artefacts.

ECSS-E-ST-40-08_1640144

p. Unresolved elements found in a Configuration artefact shall be added to UNRESOLVED_FIELD_VALUES for a later processing.

ECSS-E-ST-40-08_1640145

q. The loading and application of a Link Base artefact may be interleaved with the loading and application of the other simulator artefacts.

ECSS-E-ST-40-08_1640146

r. Unresolved elements found in a Link Base artefact shall be added to UNRESOLVED_LINKS for a later processing.

5.5.2 Simulator initialisation and configuration with one Schedule

ECSS-E-ST-40-08_1640147

a. The Activities shall be an ordered collection for which the order is the one in which they are specified in the Schedule.

ECSS-E-ST-40-08_1640148

b. The scheduling order for Events occurring at the same simulation date shall be the order in which they are specified in the Schedule.

ECSS-E-ST-40-08_1640149

c. The loading and application of a Schedule artefact may be interleaved with the loading and application of the other simulator artefacts.

ECSS-E-ST-40-08_1640150

d. Unresolved elements found in a Schedule artefact shall be added to UNRESOLVED_TASKS for a later processing.

NOTE 1. UNRESOLVED_TASKS is the name of a variable used in the implementation for the purpose of memorizing the unresolved links.

NOTE 2. Unresolved Events follow the unresolved Tasks because an Event is associated with a Task.

5.5.3 Simulator reconfiguration

ECSS-E-ST-40-08_1640151

a. During the simulator Reconnecting state, Assemblies, Link Bases, Configurations and at most one Schedule may be loaded and applied following the same sequence as specified in clause 5.5.1 and clause 5.5.2.

ECSS-E-ST-40-08_1640152

b. Conflicts found with the model hierarchy already in place shall raise an artefact loading errors.

NOTE Reconfiguration like it is defined in ECSS-E-ST-40-07 cannot delete, change or replace the existing model hierarchy. Reconfiguration can only add new models or change the current model configuration.

5.5.4 Level 2 Simulator (ISimulatorL2)

5.5.4.1 General

Errors can occur while loading a simulator artefact (i.e. Assembly, Link Base and Schedule) for the purpose of initialisation and configuration of a simulator. An error results in throwing an Invalid File Exception (see [SMPL1_FILES]). This exception has the following attribute:

- a meaningful character string providing the details that should help the simulator user to locate easily and quickly the error: useful information includes e.g. the location in the file or the line number where the error is detected, the kind of the error, the XML element which is the source of the error, etc.

The Invalid File exception inherits from *Smp::Exception* (see ECSS-E-ST-40-07) and can be used for all the Level 2 file artefacts, which are the Assembly, Schedule and Link Base.

Two types of error are possible:

- Syntax error: when the file contains a file format or syntactic error, i.e. in this case, the file is not conforming with the metadata syntax as specified in clause 5.7.
- Functional error: any error found in the file data that is not a syntax error and that prevents the correct application of the data in the simulator. For example, a non-existing model instance path or a non-existing event source or event sink is a functional error.

5.5.4.2 Requirements

ECSS-E-ST-40-08_1640153

a. The simulation environment shall provide a Simulator Object implementing the Level 2 Simulator interface as *ISimulatorL2.h* in [SMPL2_FILES].

NOTE *ISimulatorL2* extends the Level 1 *ISimulator* interface to add Level 2 simulator artefacts loading and application functionalities.

ECSS-E-ST-40-08_1640154

b. The *ISimulatorL2 LoadAssembly* method shall load and apply an Assembly file with the following arguments and behaviour:

1. Argument:
   (a) "assemblyPath" giving the path to the assembly file to load;
   (b) "parentPath" giving the path to the top-level composite to which the top-level node of the Assembly is added;
   (c) "containerName" giving the name of the container of the parent composite to add the top-level node of the assembly to;
   (d) "rootInstanceName" giving the name to give to the top-level node of the assembly. This parameter allows loading several

instances of the same assembly (with different root instance name) under the same parent.

2. Behaviour:

(a) If called while simulator is neither in the Building nor in the Standby state, it throws an InvalidSimulatorState exception as per InvalidSimulatorState.h in [SMPL1_FILES];

(b) If called with an invalid assemblyPath, it throws a FileNotFound exception as per FileNotFound.h in [SMPL1_FILES];

(c) If parentPath is empty or invalid, then the applicable parent is the Simulator, which is the composite implementing ISimulator;

(d) If the parent is the Simulator, then containerName is ignored, i.e. it is defaulted to the simulator "Models" container;

(e) If rootInstanceName is empty, then it is ignored, i.e. the name of the top-level element in the assembly is used;

(f) If the container name is not ignored and if it is invalid, it throws InvalidObjectName as per InvalidObjectName.h in [SMPL1_FILES];

(g) If the root instance name is not ignored and if it is invalid, it throws InvalidObjectName as per InvalidObjectName.h in [SMPL1_FILES];

(h) If the parent has already a child named with the given rootInstanceName, it throws DuplicateName as per DuplicateName.h in [SMPL1_FILES];

(i) Otherwise, it loads and applies the assembly file according to clause 5.5.1;

(j) If an error is encountered during the processing of the file, it throws InvalidFile as per InvalidFile.h in [SMPL1_FILES] providing a textual description of the error.

NOTE 1 The message has to contain sufficient information to identify easily and unambiguously the source element that causes the error.

NOTE 2 *LoadAssembly* can be called multiple times to load and apply multiple Assemblies in the simulator.

ECSS-E-ST-40-08_1640155

c. The *ISimulatorL2 LoadLinkBase* method shall load and apply a Link Base file with the following arguments and behaviour:

1. Argument:

(a) "linkBasePath" giving the path to the link base file to load;

(b) "parentPath" giving the path to the top-level component to which all absolute paths in the link base are relative to.

2. Behaviour:
   (a) If called while simulator is neither in the Building nor in the Standby state, it throws an InvalidSimulatorState exception as per InvalidSimulatorState.h in [SMPL1_FILES];
   (b) If called with an invalid linkBasePath, it throws a FileNotFound exception as per FileNotFound.h in [SMPL1_FILES];
   (c) If the given parentPath is empty or invalid, then the applicable parent is the Simulator, which is the composite implementing ISimulator;
   (d) Otherwise, it loads and applies the link base file according to clause 5.5.1;
   (e) If an error is encountered during the processing of the file, it throws InvalidFile as per InvalidFile.h in [SMPL1_FILES] providing a textual description of the error.

   NOTE 1 The message has to contain sufficient information to identify easily and unambiguously the source element that causes the error.

   NOTE 2 *LoadLinkBase* can be called multiple times to load and apply multiple Link Bases in the simulator.

ECSS-E-ST-40-08_1640156

d. The *ISimulatorL2 LoadSchedule* method shall load and apply a Schedule file with the following argument and behaviour:
   1. Argument:
      (a) "schedulePath" giving the path to the schedule file to load.
   2. Behaviour:
      (a) If called while simulator is neither in the Building nor in the Standby state, it throws an InvalidSimulatorState exception as per InvalidSimulatorState.h in [SMPL1_FILES];
      (b) If called with an invalid schedulePath, it throws a FileNotFound exception as per FileNotFound.h in [SMPL1_FILES];
      (c) If called more than one time, it requests the Logger to write a warning message and it returns immediately;
      (d) Otherwise, it loads and applies the schedule file according to clause 5.5.2;
      (e) If an error is encountered during the processing of the file, it throws InvalidFile as per InvalidFile.h in [SMPL1_FILES] providing a textual description of the error.

      NOTE 1 The message has to contain sufficient information to identify easily and unambiguously the source element that causes the error.

      NOTE 2 to item (c): This is because a simulator can use at most one Schedule.

ECSS-E-ST-40-08_1640157

e. The *ISimulatorL2 LoadConfiguration* method shall load and apply a Configuration file with the following arguments and behaviour:

1. Argument:
   (a) "configurationPath" giving the path to the Configuration file to load;
   (b) "parentPath" giving the path to the top-level component to which all absolute paths in the Configuration are relative to.
2. Behaviour:
   (a) If called while simulator is neither in the Building nor in the Standby state, it throws an InvalidSimulatorState exception as per InvalidSimulatorState.h in [SMPL1_FILES];
   (b) If called with an invalid configurationPath, it throws a FileNotFound exception as per FileNotFound.h in [SMPL1_FILES];
   (c) If the given parentPath is empty or invalid, then the applicable parent is the Simulator, which is the composite implementing ISimulator;
   (d) Otherwise, it loads and applies the Configuration file according to clause 5.5.1;
   (e) If an error is encountered during the processing, it throws InvalidFile as per InvalidFile.h in [SMPL1_FILES] providing a textual description of the error.

NOTE 1 The message has to contain sufficient information to identify easily and unambiguously the source element that causes the error.

NOTE 2 *LoadConfiguration* can be called multiple times to load and apply multiple Configurations in the simulator.

ECSS-E-ST-40-08_1640158

f. Unresolved elements left-over when the simulator goes from the Building state to the Connecting state shall raise an InvalidFile exception as per InvalidFile.h in [SMPL1_FILES] providing a textual description pointing to the file where the unresolved elements are found.

ECSS-E-ST-40-08_1640159

g. Unresolved elements left-over when the simulator goes from the Standby state to the Reconnecting state shall raise an InvalidFile exception as per InvalidFile.h in [SMPL1_FILES] providing a textual description pointing to the file where the unresolved elements are found.

5.6 Simulator reconfiguration

ECSS-E-ST-40-08_1640160

a. One or more Assemblies may be loaded and applied before entering the Reconnecting state.

NOTE Each corresponds to a new branch in the model hierarchy.

ECSS-E-ST-40-08_1640161

b. One or more Link Bases may be loaded and applied before entering the Reconnecting state.

ECSS-E-ST-40-08_1640162

c. One Schedule may be loaded and applied before entering the Reconnecting state.

5.7 Metadata

5.7.1 Overview

In order to make the requirements short, the generic term “Value” is used to designate a configuration element read from the Assembly. Thus, a “Value” can be one of the following:

- A Field Value
- A Property Value, i.e. a call to the corresponding Property setter
- An Operation Call
- A Global Event Subscription

The term “Link” has the same meaning as defined in Clause 5.2.7.

5.7.2 Assembly

5.7.2.1 File format specification

ECSS-E-ST-40-08_1640163

a. The Assembly file shall be in conformance with the Assembly file DRD of Annex A.

5.7.2.2 Validation rules

ECSS-E-ST-40-08_1640164

a. The Assembly XML syntax shall be compliant with Assembly.xsd.

ECSS-E-ST-40-08_1640165

b. The Assembly shall be validated at run time in a SMP compliant simulation infrastructure by calling *ISimulatorL2.LoadAssembly()*.

NOTE The Assembly can be validated offline by using the type-name form of the *Implementation* feature of model instances to navigate to the models defined in associated Catalogues.

5.7.2.3 Utilization of the Assembly

ECSS-E-ST-40-08_1640166

a. A simulator shall be configured or reconnected from at least one Assembly artefact.

NOTE One root model instance is created in the simulator corresponding to the Assembly. The model hierarchy below that root can instantiate assembly instances from additional Assembly artefacts.

ECSS-E-ST-40-08_1640167

b. A simulator may be configured or reconnected from more than one Assembly artefacts.

NOTE Each Assembly results in a different root model instance, which is the root of a new branch in the simulator model hierarchy. The model hierarchy below that root can instantiate assembly instances from additional Assembly artefacts.

ECSS-E-ST-40-08_1640168

c. Model Instances bearing Links from or to the Assembly needed for integration in a simulator shall be defined in the Assembly.

NOTE Model instances, which do not bear any Assembly external interface, can be created in C++ and thus, they do not appear in the Assembly.

ECSS-E-ST-40-08_1640169

d. Model Instances bearing Values that can be configured by the model integrator shall be defined in the Assembly.

NOTE Model instances, which do not contain any Value served to configure the simulator can be created in C++ and thus, they do not appear in the Assembly.

ECSS-E-ST-40-08_1640170

e. The model integrator shall make use of the *Implementation* feature of a Model Instance to discover the model interfaces in the documentation.

NOTE Documentation is out of the scope of ECSS-E-ST-40-08 but it can be either a SMP Catalogue or a model ICD.

ECSS-E-ST-40-08_1640171

f. A Link may specify paths to elements that do not exist at loading time and that are created dynamically during the simulator Building state or during the simulator Reconnecting state.

ECSS-E-ST-40-08_1640172

g. A Value may specify paths to elements that do not exist at loading time and that are created dynamically during the simulator Building state or during the simulator Reconnecting state.

ECSS-E-ST-40-08_1640173

h. The only simulation environment service that Operations called in the Assembly shall use is the Logger service.

5.7.3 Link Base

5.7.3.1 File format specification

ECSS-E-ST-40-08_1640174

a. The Link Base file shall be in conformance with the Link Base file DRD of Annex B.

5.7.3.2 Validation rules

ECSS-E-ST-40-08_1640175

a. The Link Base XML syntax shall be compliant with LinkBase.xsd.

ECSS-E-ST-40-08_1640176

b. The Link Base shall be validated at run time in a SMP compliant simulation infrastructure by calling *ISimulatorL2.LoadLinkBase()*.

NOTE It is impossible to perform the Link Base semantics validation offline.

5.7.3.3 Utilization of the Link Base

ECSS-E-ST-40-08_1640177

a. A simulator may be configured or reconnected from zero or several Link Base artefacts.

ECSS-E-ST-40-08_1640178

b. A Link may specify paths to elements that do not exist at loading time and that are created dynamically during the simulator Building state, during the simulator Reconnecting state and the model Configured state.

5.7.4 Schedule

5.7.4.1 File format specification

ECSS-E-ST-40-08_1640179

a. The Schedule file shall be in conformance with the Schedule file DRD of Annex C.

5.7.4.2 Validation rules

ECSS-E-ST-40-08_1640180

a. The Schedule XML syntax shall be compliant with Schedule.xsd.

ECSS-E-ST-40-08_1640181

b. The Schedule shall be validated at run time in a SMP compliant simulation infrastructure by calling *ISimulatorL2.LoadSchedule()*.

NOTE It is impossible to perform the Schedule semantics validation offline except for Execute Task Activities.

ECSS-E-ST-40-08_1640182

c. Execute Task Activities defined in a Schedule may be validated offline.

5.7.4.3 Utilization of the Schedule

ECSS-E-ST-40-08_1640183

a. A simulator shall be configured or reconnected from at most one Schedule artefact.

NOTE This unique Schedule, which a simulator is associated with, can refer to Tasks defined in additional Schedule artefacts.

ECSS-E-ST-40-08_1640184

b. The unique Schedule specified in 5.7.4.3a shall contain at least one Event.

NOTE If no events are defined, the Schedule served to configure a simulator is useless.

ECSS-E-ST-40-08_1640185

c. A simulator shall be initialised and configured from at most one Schedule artefact.

5.8 Implementation mapping

5.8.1 General

This section specifies the mapping of Assembly, Link Base and Schedule concepts to C++. This allows showing clearly the relationships between SMP Level 2 and SMP Level 1, which is specified in ECSS-E-ST-40-07.

All Assembly and LinkBase elements do have a C++ mapping.

Schedule elements do have a C++ mapping with the exception of the Transfer Activity and the Execute Task Activity.

5.8.2 Requirements

ECSS-E-ST-40-08_1640186

a. All final Paths constructed from the paths specified in the Assembly, Link Base and Schedule shall be valid Resolver service paths.

NOTE Resolver service paths are specified in ECSS-E-ST-40-07. Use of ResolveAbsolute() or ResolveRelative() to resolve paths is left to the implementation convenience.

ECSS-E-ST-40-08_1640187

b. The Model Instance shall be mapped to an instance of the C++ class defining the Model.

ECSS-E-ST-40-08_1640188

c. The Interface Link shall be created by calling AddComponent() on the Reference object of the Owner Model Instance passing the Client Model Instance as parameter.

ECSS-E-ST-40-08_1640189

d. The Event Link shall be created by calling Subscribe() on the Event Source object of the Owner Model Instance passing the Event Sink of the Client Model Instance as parameter.

ECSS-E-ST-40-08_1640190

e. The self-propagation capable Field Link shall be created by calling Connect() on the Output Field object of the Owner Model Instance passing the Input Field of the Client Model Instance as parameter.

ECSS-E-ST-40-08_1640191

f. The non-self-propagation capable (or scheduled propagation) Field Link shall be created by the simulation environment, which binds the Output Field to the Input Field using the associated published information.

ECSS-E-ST-40-08_1640192

g. The Field Value shall be mapped to the value of the corresponding Field.

ECSS-E-ST-40-08_1640193

h. The data type specified in the Field Value shall be checked against the corresponding Field data type.

ECSS-E-ST-40-08_1640194

i. The Property Value shall be mapped to a call to SetValue() on the Smp::IProperty interface.

ECSS-E-ST-40-08_1640195

j. The Operation Call shall be mapped to a call to Invoke() on the Smp::IDynamicInvocation or Smp::IOperation interface.

ECSS-E-ST-40-08_1640196

k. The Global Event Subscription shall be mapped to a call to Subscribe() on the Event Manager service.

ECSS-E-ST-40-08_1640197

l. The initialisation of the Epoch Time shall be mapped on the call to SetEpochTime() on the Smp::Services::ITimeKeeper interface.

ECSS-E-ST-40-08_1640198

m. The initialisation of the Mission Start shall be mapped on the call to SetMissionStartTime() on the Smp::Services::ITimeKeeper interface.

ECSS-E-ST-40-08_1640199

n. The Trigger Activity execution shall be mapped on the call to Execute() on the Smp::IEntryPoint interface.

ECSS-E-ST-40-08_1640200

o. The Call Operation Activity execution shall be mapped to a call to Invoke() on the Smp::IDynamicInvocation interface or on the Smp::IOperation interface.

ECSS-E-ST-40-08_1640201

p. The Set Property Activity execution shall be mapped to a call to SetValue() on the Smp::IProperty interface.

ECSS-E-ST-40-08_1640202

q. Scheduling a Simulation Time Event shall call the AddSimulationTimeEvent() on the Smp::Services::IScheduler interface.

ECSS-E-ST-40-08_1640203

r. Scheduling an Epoch Time Event shall call the AddEpochTimeEvent() on the Smp::Services::IScheduler interface.

ECSS-E-ST-40-08_1640204

s. Scheduling a Mission Time Event shall call the AddMissionTimeEvent() on the Smp::Services::IScheduler interface.

ECSS-E-ST-40-08_1640205

t. Scheduling a Zulu Time Event shall call the AddZuluTimeEvent() on the Smp::Services::IScheduler interface.

ECSS-E-ST-40-08_1640206

u. Scheduling a Global Event Triggered Event shall subscribe entry points to the Start and Stop global events by calling Subscribe() on the Smp::Services::IEventManager interface.

Annex A (normative) Assembly file - DRD

A.1 DRD identification

A.1.1 Requirement identification and source document

This DRD is called from ECSS-E-ST-40-08, clause 5.7.2.1a.

A.1.2 Purpose and objective

The purpose of the Assembly file is to contain all the metadata of the Assembly that contributes to the definition and configuration of a simulator model hierarchy.

A.2 Expected response

A.2.1 Scope and content

ECSS-E-ST-40-08_1640207

a. The suffix for assembly files shall be “smpasb”.

ECSS-E-ST-40-08_1640208

b. The document shall be compliant with the Assembly XML XSD in xml/Smdl/Assembly.xsd.

A.2.2 Special remarks

None.

Annex B (normative) Link Base file - DRD

B.1 DRD identification

B.1.1 Requirement identification and source document

This DRD is called from ECSS-E-ST-40-08, clause 5.7.3.1a.

B.1.2 Purpose and objective

The purpose of the Link Base file is to contain all the metadata of the Link Base and allows to define Links between model instances that are part of a simulator.

B.2 Expected response

B.2.1 Scope and content

ECSS-E-ST-40-08_1640209

a. The suffix for assembly files shall be “smplnk”.

ECSS-E-ST-40-08_1640210

b. The document shall be compliant with the Assembly XML XSD in xml/Smdl/LinkBase.xsd.

B.2.2 Special remarks

None.

Annex C (normative) Schedule file - DRD

C.1 DRD identification

C.1.1 Requirement identification and source document

This DRD is called from ECSS-E-ST-40-08, clause 5.7.4.1a.

C.1.2 Purpose and objective

The purpose of the Schedule file is to contain all the metadata of the Schedule allows to define Tasks and Events for the model scheduling purpose as well as new Global Events in the simulator.

C.2 Expected response

C.2.1 Scope and content

ECSS-E-ST-40-08_1640211

a. The suffix for assembly files shall be “smpsed”.

ECSS-E-ST-40-08_1640212

b. The document shall be compliant with the Assembly XML XSD in xml/Smdl/Schedule.xsd.

C.2.2 Special remarks

None.

Bibliography

ECSS-S-ST-00	ECSS system – Description, implementation and general requirements
ISO 9000 series	Quality management systems standards International Organization for Standardization (ISO) http://www.iso.org
ISO/IEC 9899:2011	ISO/IEC 9899:2011 Information technology -- Programming languages -- C
ISO/IEC 14882:2011	ISO/IEC 14882:2011 Information technology -- Programming languages -- C++
SMP v1.2	Simulation Model Portability Specification version 1.2
XML	Extensible Markup Language World Wide Web Consortium (W3C) http://www.w3.org/XML