DIN EN 16603-40-07

ICS 49.140 Einsprüche bis 2019-07-10

Entwurf

Raumfahrttechnik – Teil 40-07: Modelliersoftware-Plattform; Englische Fassung prEN 16603-40-07:2019

Space engineering – Part 40-07: Software modelling platform; English version prEN 16603-40-07:2019

Ingénierie Spatiale – Partie 40-07: Plateforme informatique de modèles de simulation; Version anglaise prEN 16603-40-07:2019

Anwendungswarnvermerk

Dieser Norm-Entwurf mit Erscheinungsdatum 2019-05-10 wird der Öffentlichkeit zur Prüfung und Stellungnahme vorgelegt.

Weil die beabsichtigte Norm von der vorliegenden Fassung abweichen kann, ist die Anwendung dieses Entwurfs besonders zu vereinbaren.

Stellungnahmen werden erbeten

- vorzugsweise online im Norm-Entwurfs-Portal von DIN unter www.din.de/go/entwuerfe bzw. für Norm-Entwürfe der DKE auch im Norm-Entwurfs-Portal der DKE unter www.entwuerfe.normenbibliothek.de, sofern dort wiedergegeben;
- oder als Datei per E-Mail an nl@din.de möglichst in Form einer Tabelle. Die Vorlage dieser Tabelle kann im Internet unter www.din.de/go/stellungnahmen-norm-entwuerfe oder für Stellungnahmen zu Norm-Entwürfen der DKE unter www.dke.de/stellungnahme abgerufen werden;
- oder in Papierform an den DIN-Normenausschuss Luft- und Raumfahrt (NL), 10772 Berlin, Saatwinkler Damm 42/43, 13627 Berlin.

Die Empfänger dieses Norm-Entwurfs werden gebeten, mit ihren Kommentaren jegliche relevanten Patentrechte, die sie kennen, mitzuteilen und unterstützende Dokumentationen zur Verfügung zu stellen.

Gesamtumfang 142 Seiten

DIN-Normenausschuss Luft- und Raumfahrt (NL)

AVA Documents Document Sharing Hub

Full document details: E DIN EN 16603-40-07:2019-06

Nationales Vorwort

Dieses Dokument (prEN 16603-40-07:2019) wurde vom Technischen Komitee CEN/CLC/JTC 5 „Raumfahrt“ erarbeitet, dessen Sekretariat von DIN (Deutschland) gehalten wird.

Das zuständige deutsche Normungsgremium ist der Arbeitsausschuss NA 131-06-02 AA „Interoperabilität von Informations-, Kommunikations- und Navigationssystemen“ im DIN-Normungsausschuss Luft- und Raumfahrt (NL).

Dieses Dokument enthält unter Berücksichtigung des DIN-Präsidialbeschlusses 1/2004 nur die englische Originalfassung von prEN 16603-40-07:2019.

Documents
Document Sharing Hub

Full document details: E DIN EN 16603-40-07:2019-06

Nationaler Anhang NA
(informativ)

Begriffe

3 Begriffe und Abkürzungen

3.1 In anderen Normen definierte Begriffe

a) Für die Anwendung dieser Norm gelten die Begriffe nach ECSS-S-ST-00-01 und ECSS-E-ST-40.

b) Für die Anwendung dieser Norm gelten die Begriffe nach ECSS-E-ST-70, insbesondere der folgende Begriff:

1) Mission

3.2 Für diese Norm spezifische Begriffe

Die im Folgenden unterstrichenen Wörter werden in demselben Abschnitt ausführlicher definiert.

3.2.1
Aggregat
Beziehung zwischen zwei Komponenten, in der eine Komponente weiterhin existieren kann, wenn die andere Komponente zerstört wird

3.2.2
Bezugspunkt
Satz von paarweisen Variablenwerten, die den Status einer Simulation definieren

3.2.3
Komponente
Baustein einer Simulation, der instanziiert werden kann und eine klar definierte Bindung mit seiner Umgebung eingeht

3.2.4
Komposit
Komponente, die eine Komposition implementiert

3.2.5
Komposition
hierarchische Beziehung, in der die untergeordnete Komponente zerstört wird, wenn die übergeordnete Komponente zerstört wird

3.2.6
Konfiguration
Festlegung von Werten für Felder von Komponenten

3.2.7
Konsument
Komponente, die entweder eine Schnittstelle aufruft oder eine Ereignisquelle subskribiert, oder die in einem ihrer Eingabefelder Daten von einem Ausgabefeld einer anderen Komponente empfängt

3.2.8
Container
typisierte Sammlung von untergeordneten Komponenten

AVA Documents
Document Sharing Hub

Full document details: E DIN EN 16603-40-07:2019-06

3.2.9
Bindung
Satz von Schnittstellen, Vorgänge, Felder, Einstiegspunkte, Ereignissenken, Ereignisquellen und alle zugehörigen Beschränkungen, die für die Interaktion mit einer Komponente verwendet werden

3.2.10
Datenübertragung
Datentransfer
Kopieren eines Werts von einem Eingabefeld in ein Ausgabefeld

3.2.11
Einstiegspunkt
Vorgang ohne Parameter, der keinen Wert ausgibt, der dem Scheduler- oder Event-Manager-Dienst zugefügt werden kann

3.2.12
Zeitraum
absolute Zeit des Simulationsszenarios

3.2.13
Event-Manager
Komponente, die die IEventManager-Schnittstelle implementiert

ANMERKUNG Die IEventManager-Schnittstelle wird in 5.3.5 festgelegt.

3.2.14
Ereignissenke
Empfänger bestimmter Benachrichtigungen, der einer Komponente gehört und über einen Subskriptions-Mechanismus registriert wird

3.2.15
Ereignisquelle
Sender bestimmter Benachrichtigungen, der einer Komponente gehört und einen Subskriptions-Mechanismus anbietet

3.2.16
Ausnahmebedingung
nicht behebbarer Fehler, der beim Aufruf eines Vorgangs oder einer Eigenschaft auftreten kann

3.2.17
Feld
Merkmal, das durch einen Wertetyp gekennzeichnet ist und einen Wert einnimmt

3.2.18
Eingabefeld
Feld, das explizit für den Empfang von Werten als Ergebnis einer Datenübertragung gekennzeichnet ist

3.2.19
Schnittstelle
benannter Satz von Eigenschaften und Vorgängen

3.2.20
Logger
Komponente, die die ILogger-Schnittstelle implementiert

ANMERKUNG Die ILogger-Schnittstelle wird in 5.3.1b festgelegt.

AVA Documents
Document Sharing Hub

Full document details: E DIN EN 16603-40-07:2019-06

3.2.21
Missionsdauer
relative Zeitmessung der verstrichenen Zeit von einem missionsspezifischen Zeitpunkt an

3.2.22
Modell
definierte Sammlung von Feldern, Zuordnungen, Einstiegspunkten, Ereignissenken, Ereignisquellen, Containern, Konstanten, Wertetypen, Eigenschaften und Vorgängen, wodurch eine Reihe von Schnittstellen realisiert wird

3.2.23
Modellimplementierung
ausführbarer Code, der ein Modell implementiert

3.2.24
Modellrealisierung
Durchführung einer Modellimplementierung

3.2.25
Ausgabefeld
Feld, das explizit als die Quelle eines Werts in einer Datenübertragung gekennzeichnet ist

3.2.26
Vorgang
Deklaration eines Verhaltensmerkmals einer Komponente oder einer Schnittstelle mit optionalen Parametern, Rückgabewert und ausgelösten Ausnahmebedingungen

3.2.27
Paket
Sammlung von Typenimplementierungen

3.2.28
Plattform
Satz von Teilsystemen/Technologien, die durch APIs und festgelegte Verwendungsmuster einen zusammenhängenden Satz von Funktionen bereitstellen

3.2.29
Eigenschaft
typisiertes Merkmal einer Klasse, einer Schnittstelle oder einer Komponente, auf das durch zwei Vorgänge, den Setter und den Getter, die nicht notwendigerweise beide vorhanden sind, zugegriffen werden kann

3.2.30
Referenz
Hinweis auf eine Komponente mit autonomem Lebenszyklus

3.2.31
Resolver
Komponente, die die IResolver-Schnittstelle implementiert

ANMERKUNG Die IResolver-Schnittstelle wird in 5.3.6 festgelegt.

3.2.32
Schedule
geordnete Ausführung von Einstiegspunkten

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

3.2.33
Scheduler
Komponente, die die IScheduler-Schnittstelle implementiert

ANMERKUNG Die IScheduler-Schnittstelle wird in 5.3.4 festgelegt.

3.2.34
Dienst
Sammlung von Feldern, Zuordnungen, Einstiegspunkten, Ereignissenken, Ereignisquellen, Schnittstellen, Konstanten, Wertetypen, Eigenschaften und Vorgängen, die anhand des Namens im globalen Bereich der Simulationsumgebung auflösbar sind

3.2.35
einfaches Feld
Feld, dessen Typ direkt einem primitiven Typen zugeordnet wird

3.2.36
Simulationsumgebung
Plattform, die die standardmäßigen E-40-07-Dienste implementiert

3.2.37
Simulationsdauer
relative Zeitdauer seit Beginn der Simulation

3.2.38
Simulator
Laufzeitkonfiguration, die in einer Simulationsumgebung ausgeführt wird

3.2.39
Simulation
Einzelausführung eines Simulators

3.2.40
Simulationsdienst
siehe Dienst

3.2.41
Quelle
Komponente, die ein oder mehrere Ausgabefelder aufweist

3.2.42
Quellkomponente
siehe Quelle

3.2.43
Zielobjekt
Komponente, die ein oder mehrere Eingabefelder aufweist

3.2.44
Zielkomponente
siehe Zielobjekt

3.2.45
Timekeeper
Komponente, die die ITimeKeeper-Schnittstelle implementiert

ANMERKUNG Die ITimeKeeper-Schnittstelle wird in 5.3.3 festgelegt.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

3.2.46
Wert
Zustand eines Wertetyps

3.2.47
Wertetyp
Menge von Werten, die eine Variable besitzen kann

3.2.48
Zulu-Zeit
die mittlere Greenwich-Zeit, auch koordinierte Weltzeit genannt

3.3 Abkürzungen

Für die Anwendung dieser Norm gelten die Abkürzungen und Symbole nach ECSS-S-ST-00-01 und die folgenden Abkürzungen und Symbole:

Abkürzung	Bedeutung
DES	Einzelereignis-Simulation (en: discrete-event simulation)
SMDL	Simulationsmodell-Definitionssprache (en: simulation model definition language)
SMP	Modelliersoftware-Plattform (en: simulation modelling platform)
URI	einheitlicher Ressourcenbezeichner (en: uniform resource identifier)
URL	einheitlicher Ressourcenzeiger (en: uniform resource locator)
UUID	universeller eindeutiger Identifikator (en: universally unique identifier)

3.4 Nomenklatur

Für dieses Dokument gilt die folgende Nomenklatur:

a) Das Wort „müssen“ (en: shall) wird in dieser Norm zur Formulierung von Anforderungen verwendet. Sämtliche Anforderungen werden durch das Wort „müssen“ ausgedrückt.

b) Das Wort „sollten“ (en: should) wird in dieser Norm zur Formulierung von Empfehlungen verwendet. Sämtliche Empfehlungen werden durch das Wort „sollten“ ausgedrückt.

ANMERKUNG Es ist zu erwarten, dass die Empfehlungen in diesem Dokument während der Anpassung entweder in Anforderungen umgewandelt oder gestrichen werden.

c) Die Wörter „dürfen“ (en: may) und „brauchen nicht“ (en: need not) werden in dieser Norm zur Formulierung der Zulässigkeit oder Unzulässigkeit verwendet. Sämtliche Zulässigkeiten werden über das Wort „dürfen“ ausgedrückt. Sämtliche Unzulässigkeiten werden über die Wörter „brauchen nicht“ ausgedrückt.

d) Das Wort „können“ (en: can) wird in dieser Norm zur Formulierung von Fähigkeiten oder Möglichkeiten verwendet und stellt daher, wenn nicht durch eines der vorstehenden Worte begleitet, einen beschreibenden Text dar.

ANMERKUNG Beim ECSS haben „dürfen“ (en: may) und „können“ (en: can) völlig unterschiedliche Bedeutungen: „dürfen“ wird normativ (Zulässigkeit) und „können“ beschreibend verwendet.

e) Zeitformen der Gegenwart und Vergangenheit werden in dieser Norm verwendet, um Tatsachenaussagen zu treffen, und stellen daher einen beschreibenden Text dar.

AVA Documents Document Sharing Hub

Full document details: E DIN EN 16603-40-07:2019-06

— Leerseite —

Documents
Document Sharing Hub

Full document details: E DIN EN 16603-40-07:2019-06

– Entwurf –

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

DRAFT
prEN 16603-40-07

May 2019

ICS 49.140

English version

Space engineering - Simulation modelling platform

Ingénierie Spatiale - Plateforme informatique de modèles de simulation

Raumfahrttechnik - Software-Modellierungs-Platform

This draft European Standard is submitted to CEN members for enquiry. It has been drawn up by the Technical Committee CEN/CLC/JTC 5.

If this draft becomes a European Standard, CEN and CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

This draft European Standard was established by CEN and CENELEC in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CEN and CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CEN and CENELEC members are the national standards bodies and national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and United Kingdom.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Warning : This document is not a European Standard. It is distributed for review and comments. It is subject to change without notice and shall not be referred to as a European Standard.

CEN-CENELEC Management Centre:
Rue de la Science 23, B-1040 Brussels

Documents
Document Sharing Hub

Full document details: E DIN EN 16603-40-07:2019-06

Table of contents

European Foreword **6**

Introduction **7**

1 Scope **8**

2 Normative references **10**

3 Terms, definitions and abbreviated terms **11**

3.1 Terms from other standards 11

3.2 Terms specific to the present standard 11

3.3 Abbreviated terms 15

3.4 Nomenclature 15

4 Principles **16**

4.1 Objectives 16

4.2 Common Concepts and common types 16

4.3 Architecture 17

4.4 Time handling principle 18

4.5 Simulation lifecycle 19

4.6 Simulation method 20

4.6.1 Discrete-event simulation (DES) 20

4.6.2 Parallelization and distribution 21

4.6.3 Inter component communication 21

4.7 Models, Services and Components 22

4.7.1 Objects 22

4.7.2 Components 23

4.7.3 Models and Services 24

4.8 Publication and Persistence 25

4.9 Dynamic invocation 26

4.10 Components meta data 28

4.10.1 Catalogue 28

4.10.2 Package 29

4.10.3 Configuration 29

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

4.11 Model exchanges considerations 29
4.11.1 SMP Bundle 29

5 Interface Requirements 30

5.1 Common 30
5.1.1 Primitive Types specification 30
5.1.2 Time Kinds 32
5.1.3 Path string 33
5.1.4 Universally Unique Identifiers (UUID) 34
5.1.5 Exception specification 34
5.2 Components and Objects interfaces 34
5.2.1 Object Specification (IObject) 34
5.2.2 Component Specification 35
5.2.3 Aggregation 38
5.2.4 Composition 41
5.2.5 Events 43
5.2.6 Entry points 45
5.2.7 Dynamic Invocation 46
5.2.8 Persistence (IPersist) 50
5.2.9 Failures 50
5.2.10 Field interfaces 51
5.2.11 Simulation Environments interface utilization 57
5.3 Simulation Environment interfaces 58
5.3.1 Service specification (IService) 58
5.3.2 Logger (ILogger interface) 58
5.3.3 Time Keeper (ITimeKeeper) 60
5.3.4 Scheduler (IScheduler) 62
5.3.5 Event Manager (IEventManager) 70
5.3.6 Resolver (IResolver) 73
5.3.7 Link Registry (ILinkRegistry) 74
5.3.8 Simulator (ISimulator) 76
5.3.9 Persistence 87
5.3.10 Publication 88
5.3.11 Type Registry 94
5.3.12 Component Factory (IFactory) 100
5.4 Meta data 101
5.4.1 Catalogue 101
5.4.2 Package 104

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

5.4.3 Configuration data 105

6 Implementation mapping 106

6.1 Catalogue to C++ 106

6.1.1 Mapping templates 106

6.1.2 Namespaces and files 109

6.1.3 Element and Type Visibility Kind 109

6.1.4 Mapping of elements 109

6.1.5 Basic Value Types 118

6.1.6 Compound Value Types 119

6.1.7 Reference Types 121

6.2 Package to library 124

6.2.1 Mapping templates 124

6.2.2 Common to Unix and Windows 125

6.2.3 Unix (Shared object) 125

6.2.4 Addendum for Windows Dynamic Link Library (DLL) 127

6.2.5 SMP Bundle 127

Annex A (normative) Catalogue DRD 128

A.1 Catalogue DRD 128

A.1.1 Requirement identification and source document 128

A.1.2 Purpose and objective 128

A.2 Expected response 128

A.2.1 Scope and content 128

A.2.2 Special remarks 128

Annex B (normative) Package DRD 129

B.1 Package DRD 129

B.1.1 Requirement identification and source document 129

B.1.2 Purpose and objective 129

B.2 Expected response 129

B.2.1 Scope and content 129

B.2.2 Special remarks 129

Annex C (normative) Configuration DRD 130

C.1 Configuration DRD 130

C.1.1 Requirement identification and source document 130

C.1.2 Purpose and objective 130

C.2 Expected response 130

C.2.1 Scope and content 130

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

C.2.2 Special remarks 130

Annex D (normative) Manifest DRD 131

D.1 Configuration DRD 131

D.1.1 Requirement identification and source document 131

D.1.2 Purpose and objective 131

D.2 Expected response 131

D.2.1 Scope and content 131

D.2.2 Special remarks 133

Bibliography 134

Figures

Figure 4-1: Common Concepts and Type System 17

Figure 4-2: SMP Architecture 17

Figure 4-3: SMP State machine 19

Figure 4-4: Object mechanisms 23

Figure 4-5: Overview of components hierarchy 23

Figure 4-6: Component Mechanisms 24

Figure 4-7: Sequence of calls for dynamic invocation 27

Tables

Table 4-1: Overview of simulation states 20

Table 4-2: ViewKind values 25

Table 5-1: Primitive Types 30

Table 5-2: Component states 36

Table 5-3: Semantically equivalent types for connections 56

Table 5-4: Default Log Message Kinds 58

Table 5-5: Condition for emitting predefined global events 72

Table 6-1: C++ declaration templates 106

Table 6-2: C++ definition templates 108

Table 6-3: C++ mapping for the Visibility kind attribute 109

Table 6-4: C++ mapping without ByPointer attribute 111

Table 6-5: C++ mapping for the Direction kind attribute 112

Table 6-6: C++ mapping without ByPointer attribute 113

Table 6-7: C++ mapping for the Operator attribute kinds 116

Table 6-8: C++ declaration templates for packages 124

Table 6-9: SMP Manifest Key 131

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

European Foreword

This document (prEN 16603-40-07:2019) has been prepared by Technical Committee CEN/CLC/TC 5 “Space”, the secretariat of which is held by DIN (Germany).

This document (prEN 16603-40-07:2019) originates from ECSS-E-ST-40-07C DIR1.

This document is currently submitted to the CEN ENQUIRY.

This document has been developed to cover specifically space systems and will the-refore have precedence over any EN covering the same scope but with a wider do-main of applicability (e.g.: aerospace).

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

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 models are often developed. This standard addresses the issues related to portability and reuse of simulation models. It is based on the work performed by ESA in the development of the Simulator Portability Standards SMP1 and 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 intended readership of this standard is the simulator software customer and supplier.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

1 Scope

ECSS-E-ST-40-07 is a standard based on ECSS-E-ST-40 for the engineering of simulation software. It includes:

- The adaptation of the ECSS-E-ST-40 requirements for simulation software, with additional specific provisions;
- The tailoring of some provisions of the ECSS-E-ST-40 requirements for simulation software;

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

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

This standard can be used:

- As an additional standard to ECSS-E-ST-40. This standard provides the additional requirements which are specific to the development of simulation software.
- To help customers in using and tailoring ECSS-E-ST-40. This standard provides the additional requirements which a customer can make applicable on the simulator software developed by the supplier.
- To assist suppliers in using ECSS-E-ST-40. This standard provides a technical specification which a supplier can follow in-order to be compliant with the requirements specific to the development of simulation software.
- To assist customers and suppliers in adapting or writing organizational procedures and standards that conform to the requirements of ECSS-E-ST-40.

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 simulation models' interfaces and environment for the purpose of model re-use and exchange to allow simulation models to be run in any conformant simulation infrastructure.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

A consequence of being compliant to this standard for a model 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 model reusability, it is only a precondition. Other characteristics of the model, to be defined outside this standard, such as its functional interfaces and behaviour, its configuration data as well as quality, suitability and performance, etc. are also heavily affecting the potential for a model to be reused. In addition agreements need to be reached on simulation infrastructure compatibility, model validation status as well as legal issues and export control restrictions.

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

Model reuse in this standard is meant both at source-code and binary level, with the latter restricted to a fixed platform.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

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.

EN reference	Reference in text	Title
EN 16601-00-01	ECSS-S-ST-00-01	ECSS system - Glossary of terms
EN 16603-40	ECSS-E-ST-40	Space engineering - Software general requirements
	https://www.w3.org/TR/xmlschema11-2/	XML schema specification
	http://www.opengroup.org	The UUID specification from Open Group.
	https://www.osgi.org/developer/specifications/	OSGi Specifications

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

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 the following term:

1. mission

3.2 Terms specific to the present standard

In the following list of terms, underlined words are further defined in the same list.

3.2.1 aggregate

relationship between two components where one component can continue to exist if the other component is destroyed

3.2.2 breakpoint

set of pairs variable-value that define the state of a simulation.

3.2.3 component

building block of a simulation that can be instantiated and that has a well-defined contract to its environment

3.2.4 composite

component implementing composition

3.2.5 composition

hierarchical relationship where child component is destroyed if the parent component is destroyed

3.2.6 configuration

specification of values for fields of components

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

3.2.7 consumer

component that either calls an interface, or subscribes to an event source, or receives data in one of its input fields from an output field of another component

3.2.8 container

typed collection of child components

3.2.9 contract

set of interfaces, operations, fields, entry points, event sinks, event sources and all the associated constraints, used to interact with a component

3.2.10 data transfer

copy of value from an input field to an output field

3.2.11 Entry Point

operation without parameters that does not return a value, which can be added to the scheduler or event manager service

3.2.12 epoch time

absolute time of the simulation scenario

3.2.13 event manager

component that implements the IEventManager interface

NOTE The IEventManager interface is specified in clause 5.3.5.

3.2.14 event sink

receiver of specific notifications, owned by a component and registered via a subscription mechanism

3.2.15 event source

emitter of specific notifications, owned by a component and offering a subscription mechanism

3.2.16 exception

non-recoverable error that can occur when calling into an operation or property

3.2.17 field

feature characterised by a value type and holding a value

3.2.18 input field

field explicitly marked for receiving values as a result of a data transfer

3.2.19 interface

named set of properties and operations

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

3.2.20 logger

component that implements the ILogger interface

NOTE The ILogger interface is specified in clause 5.3.1b.

3.2.21 mission time

relative time measuring elapsed time from a mission specific point in time

3.2.22 model

defined collection of fields, associations, entry points, event sinks, event sources, containers, constants, value types, properties and operations, realising a set of interfaces

3.2.23 model implementation

executable code implementing a model

3.2.24 model instance

occurrence of a model implementation

3.2.25 output field

field explicitly marked for being the source of a value in a data transfer

3.2.26 operation

declaration of a behavioural feature of a component or an interface with optional parameters, return value and raised exceptions

3.2.27 package

collection of type implementations

3.2.28 platform

set of subsystems/technologies that provide a coherent set of functionality through APIs and specified usage patterns

3.2.29 property

typed feature of a class, an interface or a component that can be accessed by two operations, the setter and the getter, not necessarily both present

3.2.30 reference

pointer to a component having autonomous lifecycle

3.2.31 resolver

component that implements the IResolver interface

NOTE The IResolver interface is specified in clause 5.3.6.

3.2.32 schedule

ordered execution of entry points

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

3.2.33 scheduler

component that implements the IScheduler interface

NOTE The IScheduler interface is specified in clause 5.3.4.

3.2.34 service

collection of fields, associations, entry points, event sinks, event sources, interfaces, constants, value types, properties and operations, resolvable by name in the global scope of the simulation environment

3.2.35 simple field

field of a type that maps directly to a primitive type.

3.2.36 simulation environment

platform implementing the standard E-40-07 services

3.2.37 simulation time

relative time since start of simulation

3.2.38 simulator

run-time configuration running on a simulation environment

3.2.39 simulation

single execution of a simulator

3.2.40 simulation service

See service.

3.2.41 source

component that exposes one or more output fields

3.2.42 source component

See source

3.2.43 target

component that exposes one or more input fields

3.2.44 target component

See target

3.2.45 time keeper

component that implements the ITimeKeeper interface

NOTE The ITimeKeeper interface is specified in clause 5.3.3.

3.2.46 value

state of a value type

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

3.2.47 value type

set of values which a variable can possess

3.2.48 Zulu time

the computer clock time, also called wall clock time

3.3 Abbreviated terms

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

Abbreviation	Meaning
DES	Discrete-Event Simulation
SMDL	Simulation Model Definition Language
SMP	Simulation Modelling Platform
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier

3.4 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.

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

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

4 Principles

4.1 Objectives

The main objective of this standard is to enable the effective reuse of simulation models and applications within and between space projects and their stakeholders. In particular the standard supports model reuse across different simulation environments and exchange between different organizations and missions.

The portability of models between different simulation environments is supported by defining a standard interface between the simulation environment and the models. Models can therefore be plugged into a different simulation environment without requiring any modification to the model source code.

The portability of models between different operating systems and hardware takes into consideration dependencies such as avoiding calls to operating specific APIs or use of hardware specific features. The guidelines to the model developer, on how to avoid developing models with such dependencies, is outside the scope of this standard.

4.2 Common Concepts and common types

The main purpose of SMP is to promote platform independence, interoperability and reuse of simulation models. This is done by defining;

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

In other words, models are using common concepts and a common type system to become interoperable. Thus, models 'live' in between these two common layers as shown in Figure 4-1.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

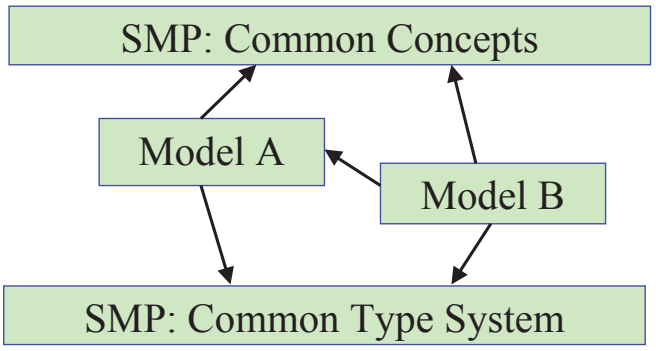

Figure 4-1: Common Concepts and Type System

4.3 Architecture

The SMP architecture covers two types of components;

- Simulation Models provide application specific behaviour;
- Simulation Environments provide Simulation Services.

This architecture is depicted in Figure 4-2.

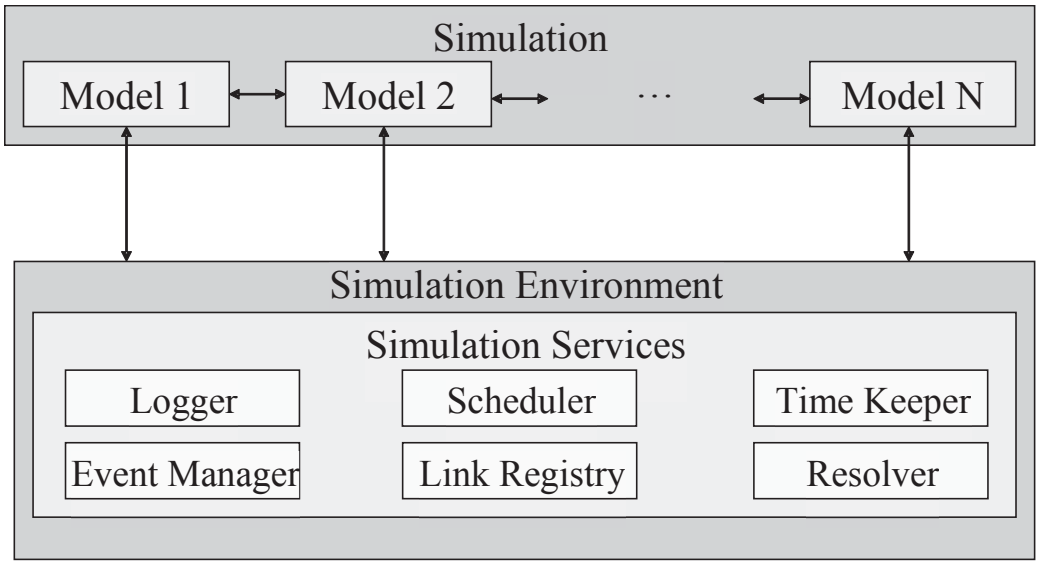

Figure 4-2: SMP Architecture

An SMP compliant simulation environment provides the following six simulation services:

- Logger: Allows logging messages (see clause 5.3.1b);
- Time Keeper: Provides the four different SMP time kinds (see clause 4.4 and 5.3.3);
- Scheduler: Allows calls of entry points based on timed or cyclic events (see clause 5.3.4);
- Event Manager: Provides mechanisms for global asynchronous events (see clause 5.3.5);

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

- Resolver: Provides the ability to get a reference to any model within a simulation (see clause 5.3.6);
- Link Registry: Maintains a list of the links between model instances (see clause 5.3.7).

In addition it supports other concepts laid out in this standard via some dedicated interfaces:

- Simulation state machine controlling interface (see clause 4.5 and 5.3.8);
- Interfaces allowing Self-persistence as described in clause 4.8 (see also IStorageReader in clause 5.3.9.1 and IStorageWriter in clause 5.3.9.2);
- Publication: A set of interfaces allowing models to publish their state to the simulation environment (see clause 5.3.10);
- Type registry: A registry allowing components to register types that later can be used for publication (see clause 5.3.11);
- Component Factory: Ability to create components via a factory (see clause 5.3.12).

The arrows in Figure 4-2 indicate interaction between components. In SMP, communication is performed via interfaces. Two different types of interfaces can be identified in this architecture:

- Interfaces between components and the Simulation Environment, and
- Inter-component communication interfaces.

4.4 Time handling principle

SMP defines four different time scales, referred to as time kinds (see clause 5.1.2 for exact specification):

- Simulation Time: Relative time since start of simulation.
- Zulu Time: Zulu Time is the computer clock time, also called wall clock time.
- Epoch Time: The absolute time of the simulation scenario.
- Mission Time: Mission time is a relative time, i.e. it measures elapsed time from a mission specific point in time.

SMP defines both Epoch and Mission Time as a fixed offset from Simulation Time. The Offset is set via calls to the SMP time keeper service and the two time kinds progress linearly with Simulation time. SMP does not define how Simulation time progress with respect to Zulu time. Typical examples of such a correlation is:

- **Real-Time:** The simulation time progresses with real-time, where real-time is typically defined by the computer clock.
- **Accelerated**: The simulation time progresses relative to real-time using a constant acceleration factor. This factor can be larger than 1.0, which relates to "faster than real-time", smaller than 1.0, which means "slower than real-time", or 1.0, which coincides with real-time.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

- **Free Running**: The simulation time progresses as fast as possible, and is not related to real-time. Typically, the speed is coordinated with the timed events of the scheduler, which underlines the close relationship between these two services (Time Keeper and Scheduler).
- **Debugging**: The simulation is executed in a step-by-step manner using break points in order to inspect data or trace calls within the simulation.

SMP does not mandate which of these modes a simulation environment supports.

4.5 Simulation lifecycle

Any SMP simulation goes via a lifecycle as defined in Figure 4-3. The simulation environment is responsible to ensure that this state diagram is followed. It is controlled via the ISimulator interface (see clause 5.3.8).

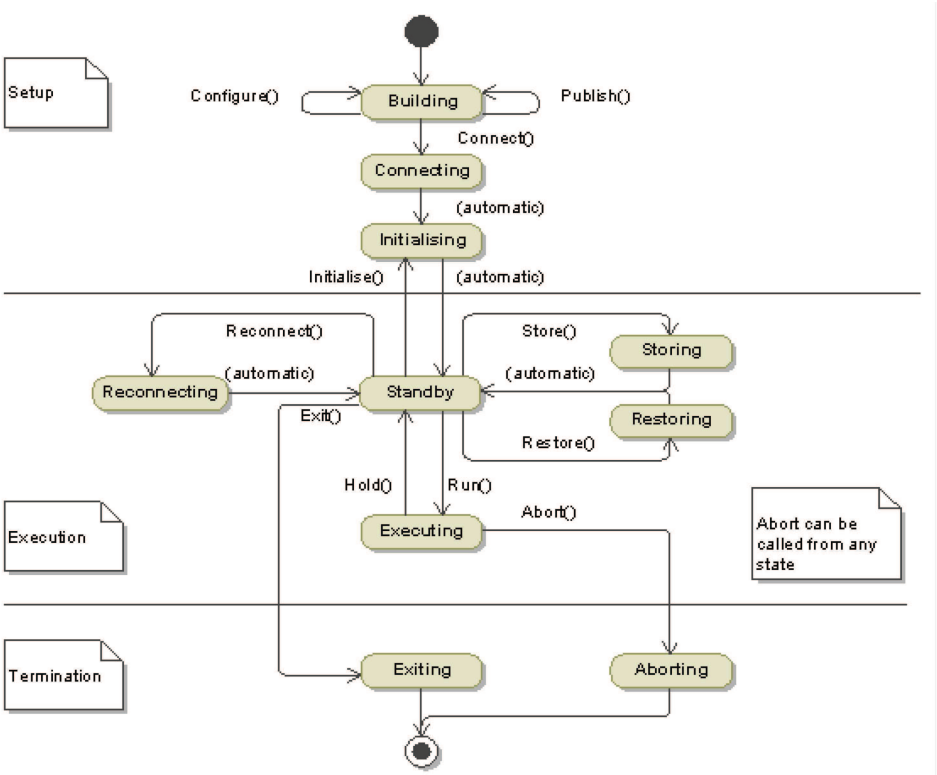

Figure 4-3: SMP State machine

Each state in Figure 4-3 has its own purpose and behaviour as explained in Table 4-1. Notice that some state transitions are automatically performed by the Simulation Environment as indicated in Figure 4-3, while others need explicit calls to the ISimulator interface.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

Table 4-1: Overview of simulation states

Name	Description
Building	In Building state, the model hierarchy is created.
Connecting	In Connecting state, the simulation environment traverses the model hierarchy and calls the Connect() method of each model.
Initialising	In Initialising state, the simulation environment executes all initialization entry points in the order they have been added to the simulator using the ISimulator AddInitEntryPoint()method (see 5.3.8).
Standby	The simulation environment does not progress simulation time. Only entry points registered relative to Zulu time are executed.
Executing	The simulation environment does progress simulation time. Entry points registered with any of the available time kinds are executed.
Storing	In Storing state, the simulation environment first stores the values of all fields published with the State attribute to storage (typically a file). Afterwards, the Store() method of all components (Models and Services) implementing the optional IPersist interface is called, to allow custom storing of additional information.
Restoring	In Restoring state, the simulation environment first restores the values of all fields published with the State attribute from storage. Afterwards, the Restore() method of all components implementing the optional IPersist interface is called, to allow custom restoring of additional information.
Reconnecting	In Reconnecting state, the simulation environment makes sure that models that have been added to the simulator after leaving the Building state are properly published, configured and connected.
Exiting	In Exiting state, the simulation environment is properly terminating a running simulation.
Aborting	In this state, the simulation environment attempts a simulation shut-down, whereby the simulation can stop executing as the users expect, without guaranties for actual release of resources.

4.6 Simulation method

4.6.1 Discrete-event simulation (DES)

SMP is built on discrete-event simulation (DES) theory where the behaviour of a system is modelled as a discrete sequence of events in time. Each event that marks a change in the state of the systems occurs at a particular instant in time. The simulation can jump in time from one event to the next since no change in the system occurs between consecutive events.

The main elements in SMP that support this approach are:

- The simulation components schedule EntryPoints (see clause 5.2.6.1) on the SMP scheduler (see clause 5.3.4) for execution of events.
- The Simulation state is captured in the persisted data.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

4.6.2 Parallelization and distribution

SMP assumes a single scheduler executing its events in sequence. All components are loaded inside the same address space allowing direct communication between them. The standard however does not prevent parallelization or distribution to be built into layers on top of the standard.

4.6.3 Inter component communication

4.6.3.1 Overview

SMP supports the following main method of communication between components:

- Direct interface based communication
- Data flow based communication
- Event based communication

4.6.3.2 Interface based communication

An interface-based design adds interfaces as the standard mechanisms for inter-model communication. This isolates the definition of an interface (the "contract") from its implementation. In an interface-based design, a model provides any number of interfaces. An Interface defines a contract between models. Every model implementing the interface provides all the functionality of the interface, so that every model, which consumes this interface can rely on the interface implementation. As interfaces are a mechanism to de-couple models, they do not give access to fields, but only to operations. With special operations (i.e. use of Properties. See definition 3.2.29 "property") that read or write a single value, access to fields can be added.

4.6.3.3 Data flow based communication

4.6.3.3.1 Overview

In a data flow based communication, two components exchange the value of a field. The Provider component transfers an Output field value into an Input field of the Consumer component. The Output field is said to be connected to the Input field through a Dataflow connection.

The dataflow communication can be automatic, i.e. whenever the Output field is updated by its owning component, the value is immediately propagated to the connected Input field. The output fields which take part in an automatic dataflow based communication implement IDataflowField (see 5.2.10.7).

The dataflow communication can be scheduled, i.e. when the Output field is updated by its owning component, the value is not automatically propagated to the connected Input field. The transfer is only performed on request, e.g. cyclically by the Simulation Infrastructure.

It is neither mandated when an Output field pushes its value into connected Input fields, nor whether an Input field performs any specific action after it has been updated. An Input field can be implemented in a way that it notifies its

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

containing model about a change, which can be used to trigger certain behaviour in the Consumer component.

4.6.3.3.2 Data types consideration

When two fields are involved in a dataflow connection, compatibility of the field types ensures correct transfer of data between the provider model and the consumer model.

Two levels of type compatibility are specified for the fields:

a. Strict compatibility: both fields are typed by the same type as identified by their type UUID published in the Type Registry. In this case, it is obvious that the fields can be connected via a dataflow connection.

b. Equivalence: the types are equivalent as per their semantics or physical representation. For example, when type 1 is a user-defined type Voltage mapped on a Smp::Float64 and type 2 is an user-defined type Tension mapped on Smp::Float64. As both type 1 and type 2 are Smp::Float64, they are said “equivalent”. Dataflow connection is allowed as there is no issue to transfer information between type 1 and type 2.

4.6.3.4 Event Based communication

For an event-based design the components are modelled using Event Sinks (see 5.2.5.1) and Event Sources (see 5.2.5.2). Events issued by the source are received by all sinks registered to receive the corresponding event.

4.7 Models, Services and Components

4.7.1 Objects

An object is the base class for all SMP elements. It provides the basic features of a name, description and parent to all SMP elements. This implies that all SMP elements are organized in a hierarchical structure and always able to traverse upwards towards its root. The elements inheriting directly from Object are:

- Entry Points (IEntryPoint) as void operations that can be called by the scheduler and event manager services;
- Entry Point Publisher (IEntryPointPublisher) for publishing of entry points;
- Event Sinks (IEventSink) and Event Sources (IEventSource) for event based communication between objects;
- Fields (IField) to hold the simulation state and data;
- Composites (IComposite) and Containers (IContainer) to build the object hierarchy;
- References (IReference) allowing objects reference other components;
- Failures (IFailure) to allow objects to represent a failure in a system;
- Components (IComponent) implementing the simulation behaviour;

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

- Types (IType) to be used for the definition of fields;
- Factory (IFactory) that creates components;
- Persistent Objects (IPersist) that can store and restore their state.

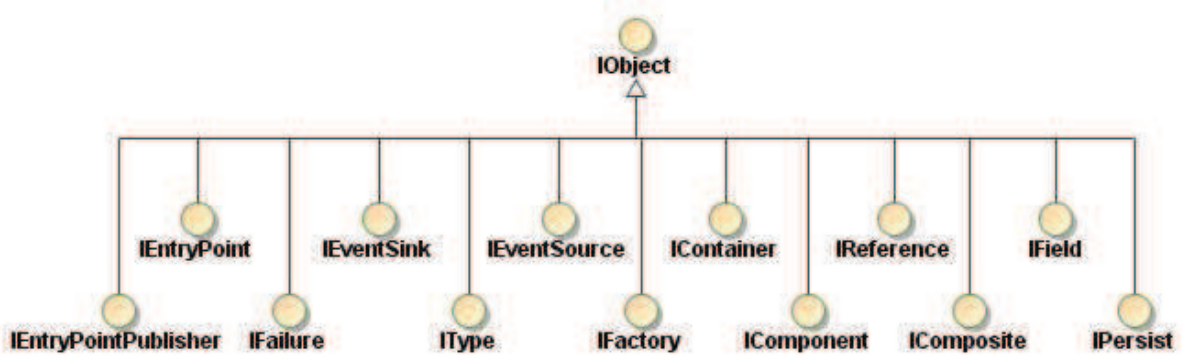

Figure 4-4: Object mechanisms

4.7.2 Components

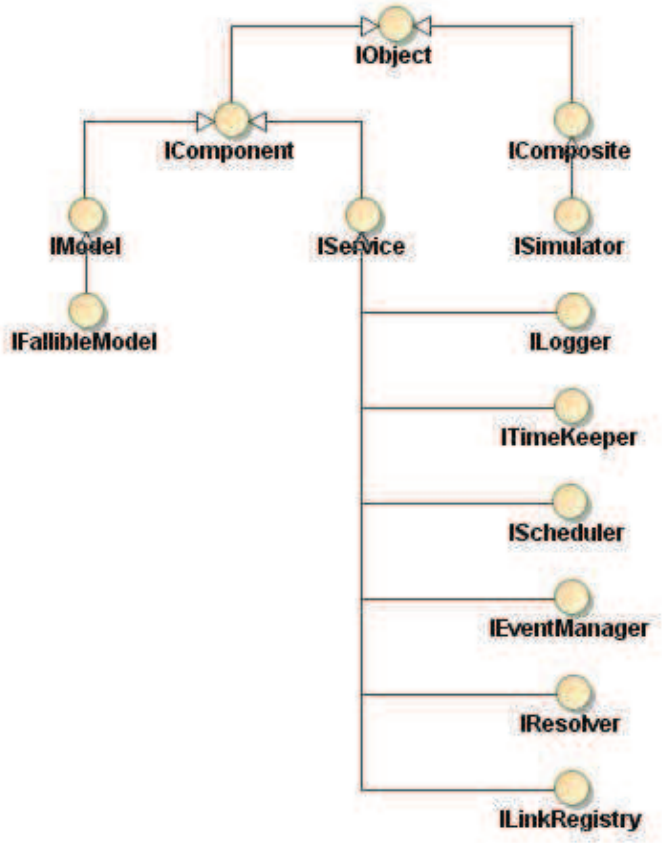

Figure 4-5: Overview of components hierarchy

The functionality of an SMP based simulator is implemented in elements that implement the IComponent (see 5.2.2.1) interface. A component represents an implementation of a self-contained feature with well-defined interfaces to other

AVA Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

components. At initialization time, a simulation is built by assembling a set of instances of components. In addition, to implementing the IComponent interface, a number of additional optional component mechanisms are specified (see Figure 4-6):

- Component aggregation (IAggregate),
- Inter-component events (IEventProvider and IEventConsumer),
- Dynamic invocation (IDynamicInvocation),
- Link management features (ILinkingComponent),

See clause 5.2 for details on Component Mechanisms.

The Simulator itself is a component, in particular a composite. All its direct children are components, namely models and services.

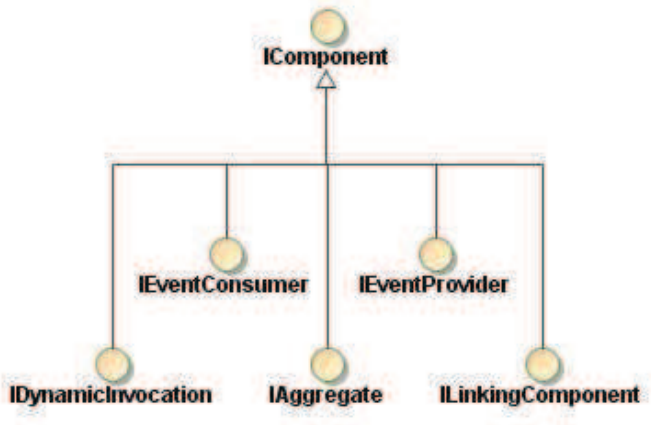

Figure 4-6: Component Mechanisms

4.7.3 Models and Services

Two main flavours of components are specified by SMP: Models and Services. The main differences are:

- Models implement the IModel interface while Services implement the IServices interface. Both interfaces are empty and do not add any additional capabilities, but the difference allows to efficiently differentiate models and services.
- Models are added to the simulation in a hierarchical tree, while services live in the global scope of the simulation.
- The Models can be fallible by implementing the IFallibleModel interface but Services are not.
- Models are added to the Simulations via the ISimulator AddModel method, while Services are added via the AddService method.
- It is possible to get a pointer to a service from the ISimulator interface via the GetService method by its name. This implies that all components in a simulation can easily obtain a reference to a service.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

The mandatory features of an SMP runtime environment are specified as services. See clause 5.3 and Figure 4-5.

4.8 Publication and Persistence

SMP components publish their state information to the simulation environment to:

- Allow visualization of the simulation state.
- Allow the simulation environment to interact with the state of the component.
- Allow the simulation environment to store and restore the state of the component via the SMP persistence mechanism.

All published fields are annotated with a set of attributes provided by the SMP component to the Simulation Environment:

- A View Kind attribute indicating which kind of user this information is intended for. The values and intended interpretation of these values by the Simulation Environment is given in Table 4-2.
- If the field is part of the breakpoint or not (State attribute of field).
- If it is an Input, or an Output, or an Input/Output field (Input and Output attributes of field).

Additional meta information can also be provided via the SMP Catalogue (see 5.4.1).

Table 4-2: ViewKind values

Name	Intended interpretation
VK_None	The element is not made visible to the user.
VK_Debug	The element is made visible for debugging purposes. The element is not visible to end users. If the simulation infrastructure supports the selection of different user roles, then the element is intended to be visible to "Debug" users only.
VK_Expert	The element is made visible for expert users. The element is not visible to end users. If the simulation infrastructure supports the selection of different user roles, then the element is intended to be visible to "Debug" and "Expert" users.
VK_All	The element is made visible to all users. (this is the default)

From the list of published fields, the simulation environment is able to determine the state of a simulation and store it into a breakpoint (or to restore it when needed). This is called persistence. Persistence of SMP components can be handled in one of two ways:

- **External Persistence**: The simulation environment stores and restores the model's state by directly accessing the fields that are published to the simulation environment, i.e. via the IPublication (See 5.3.10.1) interface.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

- **Self-Persistence**: The component can implement the IPersist (See 5.2.8) interface, which allows it to perform special operations during store and restore in addition to external persistence. Typically, self-persistence allows the persistence of dynamic data structure (e.g. events on the simulation schedule). Two approaches exist in this case for models to store their data:
  - Its state or parts of it can be stored/restored in the storage that is provided by the simulation environment via the IStorageReader (see 5.3.9.1) and IStorageWriter (see 5.3.9.2) interfaces provided by the simulation environment.
  - The component can query the filename and location of the storage file from the environment via the IStorageReader (see 5.3.9.1) and IStorageWriter (see 5.3.9.2) interfaces and store additional files in the same location. This mechanism is usually only needed by specialised models, for example embedded models that need to load on-board software from a specific file.

SMP Runtime Environments supports both External and Self-Persistence. For models and components, only external persistence (via the Store() and Restore() methods of the ISimulator interface) is a mandatory feature, while self-persistence is an additional optional mechanism.

4.9 Dynamic invocation

SMP supports dynamic invocation allowing interaction between simulation environments and simulation models. This is typically used during execution allowing to control a simulation via scripting. It is a mechanism that makes the operations of a component available via a standardised interface.

In order to allow calling a named method with any number of parameters, a request object is created which contains all information for the method invocation. This request object is also used to transfer back a return value. The dynamic invocation concept standardises the request objects (IRequest interface, see 5.2.7.2). In addition, two methods are provided as part of IDynamicInvocation to create and delete request objects. However, it is not mandatory to use these methods, as request objects can as well be created and deleted using another implementation. A reason for doing this could be to minimise the number of round-trips between a client (that calls a method) and a component that implements IDynamicInvocation. The sequence diagram in Figure 4-7 shows all steps involved when using the CreateRequest() and https://myteams.ecss.nl/ta/DeleteRequest() methods.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

Figure 4-7: Sequence of calls for dynamic invocation

The sequence diagram in Figure 4-7, using a Client component and a Model implementing IDynamicInvocation, contains the following steps:

1. The client calls the CreateRequest() operation of the component to create a request object for the operation, passing it the name of the operation.
2. The component creates a request object for the operation, using the default values of all parameters.
3. The component returns the Request object via its IRequest interface to the client.
4. The client calls the SetParameterValue() operation of the Request object to set parameters to non-default values.
5. The client calls the Invoke() operation of the component to invoke the corresponding operation.
6. The component calls the GetParameterValue() operation of the Request object to get parameters.
7. The component calls its internal operation that corresponds to the invoked operation.
8. The component calls the SetReturnValue() operation of the Request object to set the return value.
9. The component returns control to the client.
10. The client calls the GetReturnValue() operation of the Request object to get the return value.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

11. The client calls the DeleteRequest() operation of the component to delete the Request object.
12. The component destroys the request object.
13. The component returns control to the client.

4.10 Components meta data

4.10.1 Catalogue

Meta data for SMP objects are stored in XML documents called the Catalogue. Having the SMP objects described in XML catalogues allows taking benefit from the XML language, for example:

1. Generation of the catalogues from UML diagrams
2. Generation of models documentation from the catalogue
3. Generation of models skeleton code from the catalogue (See clause 6.1).

The content of a catalogue is hierarchically ordered in namespaces that may be nested. Inside each namespace many uniquely named instance of the following SMP features can be found:

- Types definitions including:
  - Constants, Fields and Properties
  - Exceptions
  - Data Types
- Interfaces specifications
- Component and model specifications including:
  - Event Sinks and sources
  - Fields and properties
  - Entry Points
  - Operations
  - Containment and inheritance
  - Interfaces and references
- Attributes that can be attached to elements:
  - Fallible and Forcible
  - Min and Max limits for types
  - View/ViewKind information determines the visibility of the element.

For all the elements above, meta data can be added like the description of each element or the engineering unit for type definitions. From this, it can be seen that the Catalogue definition provides a rich capability to describe the complete external interface of all SMP components. In fact, the interfaces as described in the SMP standard can as well be expressed in a catalogue. (See ecss.smp.smpcat as referenced in clause 5.4.1.2.1a)

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

4.10.2 Package

A package describes how implementations of types defined in catalogues are packaged. This includes not only models, which may have different implementations in different packages, but as well all other user-defined types.

4.10.3 Configuration

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

4.11 Model exchanges considerations

One of the primary goal of SMP is to allow model exchanges based on the Package concept.

Model source code exchange are considered easier than binary exchange as some considerations are important to be taken into account when exchanging binary models.

The mapping of a Package to C++ defines which symbols a static or dynamic library of SMP has to expose. This enables binary distribution of models, where only the catalogues and/or header files (for the compiler) and the libraries (for the linker) are provided, but no implementation source code. Nevertheless, binary compatibility depends on a number of other constraints, which may even vary between operating systems and compilers.

4.11.1 SMP Bundle

For distribution of a binary package SMP bundles are used. A SMP Bundle is an archive (e.g. a tar file on Linux, or a zip file on Windows) which provides the following elements:

- One or more SMDL packages.
- One or more package dynamic libraries, directly related to the SMDL packages.
- One or more package static libraries, directly related to the SMDL packages.
- All the SMP catalogues related to the SMDL packages.
- Optionally include other artefacts (SMDL configurations) and/or the related source code for all or parts of the included SMDL packages.

The related structure of folders and files within the bundle, and the names of folders and files are not standardised.

The added value of a Bundle is the additional SMP.MF Bundle Manifest file.

This Manifest is an ASCII file (aligned with the OSGi bundle manifest format) which contains key-value pairs with important meta data for the bundle.

AVA Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

5 Interface Requirements

5.1 Common

5.1.1 Primitive Types specification

a. All SMP fields, parameters, constants and properties shall be of either a Primitive Type as per PrimitiveTypes.h in [ZIPFILE], or a type derived from one or more of these primitive types.

NOTE This specification is compliant with the types specified in Table 5-1.

b. Mapping between SMP types, XML types and ISO/ANSI C++ types shall be as per Table 5-1.

NOTE C++ mapping for primitive types is provided by PrimitiveTypes.h in [ZIPFILE].

Table 5-1: Primitive Types

SMP Type	XML mapping	C++ mapping	Description
Char8	xsd:string	char	8 bit character type to represent textual characters
String8	xsd:string	const char*	8-bit character strings based on UTF-8 encoding, which is commonly used in XML
Bool	xsd:boolean	bool	Bool is a binary logical type with values true or false
Int8	xsd:byte	int8_t	8 bit signed integer
UInt8	xsd:unsignedByte	uint8_t	8 bit unsigned integer
Int16	xsd:short	int16_t	16 bit signed integer
UInt16	xsd:unsignedShort	uint16_t	16 bit unsigned integer
Int32	xsd:int	int32_t	32 bit signed integer
Unit32	xsd:unsignedInt	uint32_t	32 bit unsigned integer
Int64	xsd:long	int64_t	64 bit signed integer
UInt64	xsd:unsignedLong	uint64_t	64 bit unsigned integer

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

SMP Type	XML mapping	C++ mapping	Description
Float32	xsd:float	float	IEEE 754 single-precision floating-point type with a length of 32 bits.
Float64	xsd:double	double	IEEE 754 double-precision floating-point type with a length of 64 bits.
Duration	xsd:duration	int64_t	Duration in nanoseconds. See 5.1.1c for detailed specification.
DateTime	xsd:dateTime	int64_t	Absolute time in nanoseconds. See 5.1.1d for detailed specification

c. The Duration type as per Table 5-1 shall be used for specifying a duration, as follows:

1. It is expressed in nanoseconds;
2. It is stored in a signed 64 bit integer;
3. Positive values correspond to positive durations;
4. Negative values correspond to negative durations.

NOTE 1 Nanoseconds is the lowest level of granularity supported for time in SMP.

NOTE 2 The duration type allows specifying values roughly between -290 years and 290 years.

NOTE 3 The duration type allows expression of relative time, hence "negative duration" implies a relative time in the past.

d. The DateTime type as per Table 5-1 shall be used for absolute time values, as follows:

1. It is expressed in nanoseconds, relative to the reference time of 01.01.2000, 12:00, Modified Julian Date (MJD) 2000+0.5;
2. It is stored in a signed 64 bit integer;
3. Positive values correspond to times after the reference time;
4. Negative values correspond to time values before the reference time.

NOTE 1 Nanoseconds is the lowest level of granularity supported for time in SMP.

NOTE 2 DateTime allows specifying time values roughly between 1710 and 2290.

e. A SMP Simple Field shall be of a type that maps directly to a Primitive Type.

f. The AnySimple type shall hold a Primitive Type as per AnySimple.h in [ZIPFILE].

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

g. The AnySimpleArray type shall be an array of AnySimples as per AnySimpleArray in [ZIPFILE].

5.1.2 Time Kinds

a. Simulation time shall be used for keeping the progress of time with respect to the start of the simulation, with the following properties:
   1. Simulation time is a non-negative Duration type;
   2. Simulation time is initialised to 0 at the beginning of the Setup phase;
   3. Simulation time progresses only when the simulation environment is in Executing state;
   4. It is not specified how quickly simulation time is progressed when the simulator is in Executing state;
   5. Simulation time is stored and restored in the storing and restoring states;

b. Mission time shall be used for keeping the progress of relative time with respect to a Mission Start time, with the following properties:
   1. Mission time is initialised to 0 at the beginning of the Setup phase;
   2. Mission Time is calculated as a fixed offset between the current Epoch time and the given Mission start time according to the following formula: MissionTime = EpochTime – MissionStartTime;
   3. The Mission time progresses with Epoch time, which progresses with Simulation time, and is hence affected by the ITimeKeeper SetEpochTime method.
   4. The Mission time offset from Epoch time, the Mission start time changes by calls to:
      (a) the ITimeKeeper SetMissionTime method;
      (b) the ITimeKeeper SetMissionStartTime method.
   5. Mission time only progresses when the simulation environment is in Executing state;
   6. Mission time is stored and restored in the storing and restoring states;
   7. Mission time is stored as a Duration type.

c. Zulu time shall be time dependent on the system clock of the host machine expressed using the DateTime type.

d. Epoch time shall be time dependent on the Simulation time with a fixed offset using the DateTime type.

NOTE 1 Epoch time progresses with Simulation time.

NOTE 2 Epoch time is changed with the ITimeKeeper SetEpochTime (See 5.3.3).

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

5.1.3 Path string

a. An SMP path string shall be a representation of a valid route from an SMP object in the hierarchy to another SMP object.

NOTE 1 Examples of valid path strings:

- /Satellite/Receivers/Receiver1
- /Logger
- /Logger/
- ../../Transmitters/Transmitter4
- ./Satellite/../Satellite//Receivers/

NOTE 2 Examples of invalid path strings:

- "/..", parent of root object do not exist
- "…", meaning of triple dots not defined.

b. Both Absolute and Relative path strings shall be supported and distinguished as follows:

1. Paths starting with a delimiter are absolute paths from the simulation root object.
2. Paths not starting with a delimiter are relative paths from the current object.

c. The delimiter between component names in the path string shall be "/".

d. The delimiters between components and its children objects that are not components shall be either "/" or ".".

NOTE This allows "Component.Operation()" to be used as path.

e. Trailing delimiters shall be allowed in path strings.

f. It shall be possible to reference the parent object by the ".." string.

g. It shall be possible to reference the current object by:

1. the "." string
2. an empty string ""

NOTE This allows the following to be used as path to operations of current object:

- .Operation()
- ./Operation()

h. The path string shall allow an element in an array to be identified by "[n]" trailing the array name where "n" is the zero based element index, with no delimiter.

NOTE This allows the following to be used for addressing element 2 of an array "MyArray" in MyModel:

- MyModel/MyArray[2]
- MyModel.MyArray[2]

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

5.1.4 Universally Unique Identifiers (UUID)

a. All SMP types shall have a unique UUID as per Uuid.h in [ZIPFILE].

NOTE 1 The UUID follows the specification from Open Group (http://pubs.opengroup.org/onlinepubs/9629399/apdxa.htm)

NOTE 2 The UUID is a 128 bit long unique identifier.

NOTE 3 The UUID allows for example to:

- Uniquely identify types defined in catalogues so that can be bound with implementations defined within packages.
- Uniquely identify linked elements within a Catalogue.

5.1.5 Exception specification

a. All SMP exceptions shall inherit from the Exception class as per Exception.h in [ZIPFILE] providing the following information:

1. The description of the exception;
2. The name of the exception;
3. The exception message.

NOTE This covers both exceptions defined in this standard and user defined exceptions.

5.2 Components and Objects interfaces

5.2.1 Object Specification (IObject)

a. All SMP objects shall provide the following features as per IObject.h in [ZIPFILE]:

1. If the object is not an array element, a name of the object as follows:
   (a) Not be empty;
   (b) Start with a letter;
   (c) Contain only letters, digits, the underscore ("_") and brackets ("[" and "]");
   (d) Not be an ISO/ANSI C++ keyword.
2. If the object is an array element, a name identical to the zero based element index;
3. A description of the object;
4. The parent object as follows:
   (a) An IObject pointer to the parent if the object has a parent;
   (b) A NULL pointer if the object does not have a parent.

NOTE 1 The Object description may be empty.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06

NOTE 2 All SMP elements inherit from the IObject interface including:

- Entry Points
- Event Sinks and Sources
- Fields
- Containers
- References
- Failures
- Components
- Composites
- Collections
- Factories
- Types

NOTE 3 to item 5.2.1a.1(d): See ISO/IEC 9899:2011 [C11 Standard] and ISO/IEC 14882:2011 [C++11 Standard] for the actual list of keywords.

b. All SMP objects with the same parent shall have a unique name.

c. The validity of the SMP name shall be checked when an SMP object is created, with the following behaviour:

1. If an object with an invalid name is created, it throws a InvalidObjectName exception as per InvalidObjectName.h in [ZIPFILE].

5.2.2 Component Specification

5.2.2.1 Component (IComponent)

a. All SMP Components shall implement the IComponent interface as per IComponent.h in [ZIPFILE].

b. The IComponent GetState method shall return the current state of the component as per ComponentStateKind.h in [ZIPFILE], specified in Table 5-2.

Documents
Document Sharing Hub
Full document details: E DIN EN 16603-40-07:2019-06